МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Кафедра прикладної математики

КУРСОВА РОБОТА
з дисципліни «Програмування»

на тему:

АВЛ-дерево

Виконала:
студентка І курсу
спеціальність 113 КМ-03
Прикладна математика
ПЮСТОНЕН С.Р.

Керівниця:
Любашенко Н.Д.

Національна оцінка:

_____

Кількість балів:____

Київ-2021

ЗМІСТ

ВСТУП

Мета роботи: поглиблення та поширення теоретичних знань про можливості мови програмування С, засвоєння теоретичного матеріалу та набуття початкових практичних навичок при проектуванні та реалізації програмного продукту.

Об’єкт дослідження: АВЛ-дерево.

Предмет дослідження: реалізація АВЛ-дерева та операцій додавання, видалення, пошук елементів АВЛ-дерева.

1. Постановка задачі

Нижче буде наведено теоретичні відомості про об’єкт дослідження.

Теоретичні відомості про об’єкт дослідження

З розвитком комп'ютерної техніки проблема зберігання та обробки великих об'єктів даних становилася все більш актуальною. Виникла необхідність організації *сховищ* для таких об'єктів даних, які пропонують можливість швидко знаходити та модифікувати дані. Один із способів організації такого зберігання – АВЛ-дерево.

АВЛ-Дерево (англ. AVL Tree) — збалансоване бінарне дерево, різниця між висотою лівого та правого піддерева для кожної з вершин якого не більше одиниці. АВЛ-дерево також називають збалансованим за висотою бінарним деревом. Назва походить від ініціалів винахідників Г. М. Адельсон-Вельский (англ. G.M. Adelson-Velskii) та Е. М. Ландіс (англ. E.M. Landis), які описали цю структуру даних ще в 1962 році. На Рис.1.1 зображено приклад збалансованого АВЛ-дерева.

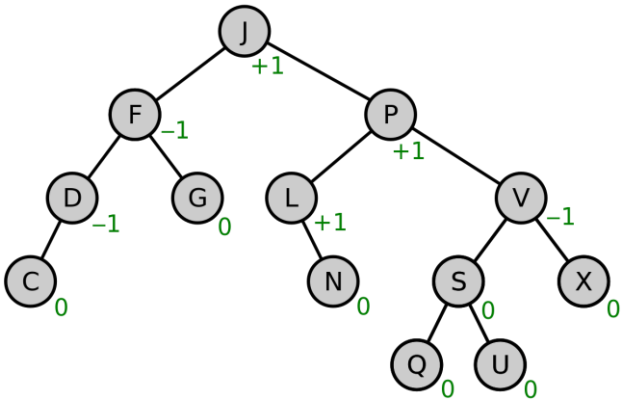

Рис.1.1 Зразок збалансованого AVL-дерева

Балансування. Балансуванням вершини називається операція, яка в разі різниці висот лівого і правого піддерев (balance > 1 або balance <-1), змінює зв'язки предок-нащадок у піддереві даної вершини так, що різниця стає -1≤ balance ≤1. Балансування здійснюється шляхом обертань піддерева даної вершини. Використовуються 4 типи обертань(приклади у візуалізаторі реалізовані за допомогою[3]):

1. ***Мале ліве обертання.*** Balance(a)<-1 & balance(b)≤0: обертання використовують тоді коли характеристика вузла a становиться менше -1 (balance (a)<-1) тобто висота піддерева b опинилась на 2 більше висоти піддерева L і при цьому висота піддерева C не більша висоти R тобто balance(b)≤0. Рис 1.2. Рис.1.3.

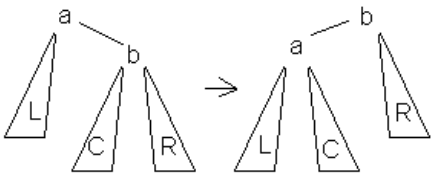

Рис.1.2 Мале ліве обертання

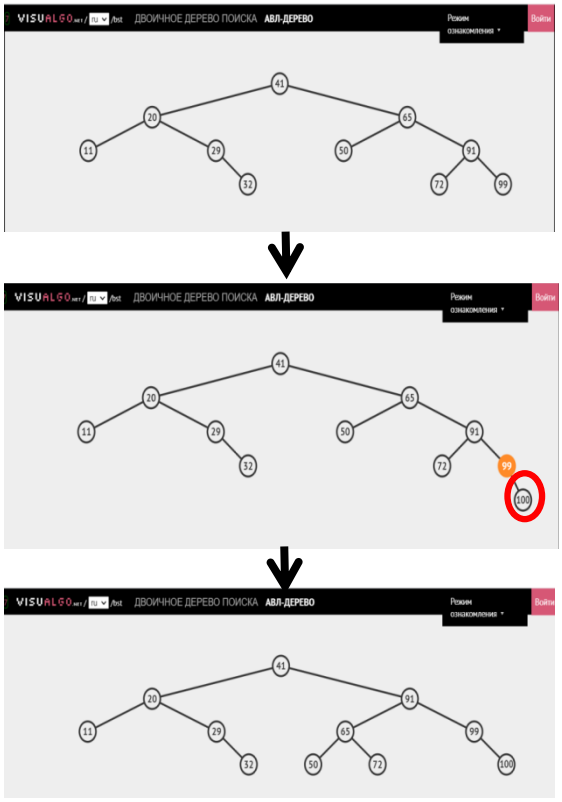

Рис1.3 Мале ліве обертання(виконано у візуалізаторі)

2. ***Велике ліве обертання.*** Balance(a)<-1 & balance(b)>0: виконується тоді, коли висота піддерева b стає на 2 більше висоти піддерева L і 12при цьому висота піддерева C більша висоти R тобто balance(b)>0. Рис.1.4. Рис.1.5.

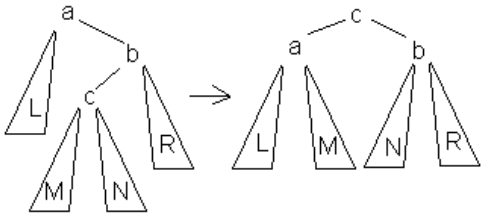

Рис.1.4 Велике ліве обертання

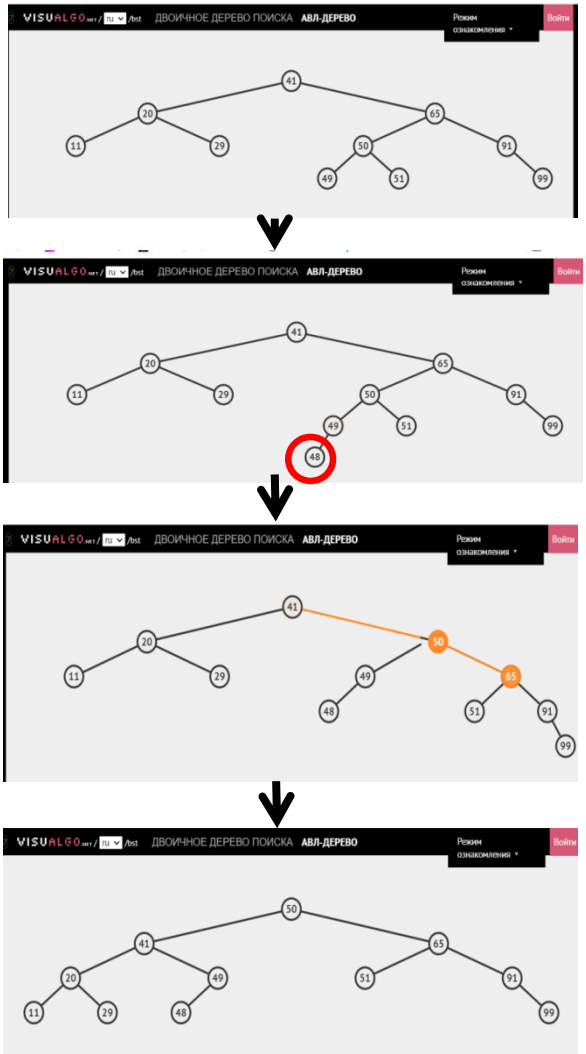

Рис.1.5 Велике ліве обертання(виконано у візуалізаторі)

3. ***Мале праве обертання***. Balance(a)>1 & balance(b)≥0: виконується тоді, коли висота піддерева b стає на 2 більше висоти піддерева R і при цьому висота піддерева C не більша висоти L тобто balance(b)≥0. Рис.1.6. Рис.1.7.

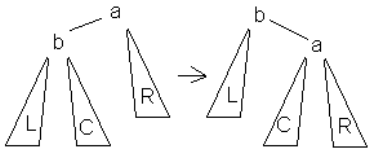

Рис.1.6 Мале праве обертання

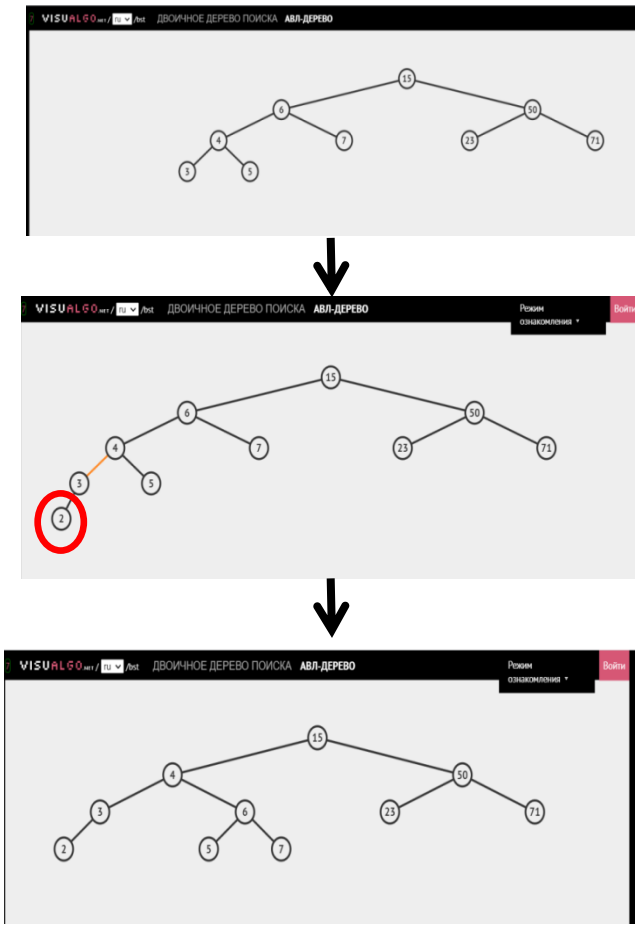

Рис.1.7 Мале праве обертання(виконано у візуалізаторі)

4. ***Велике праве обертання.*** Balance(a)>1 & balance(b)<0: виконується тоді, коли висота піддерева b стає на 2 більше висоти піддерева R і при цьому висота піддерева C не більша висоти L тобто balance (b)<0. Рис.1.8. Рис.1.9.

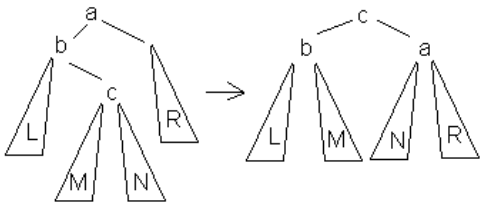

Рис.1.8 Велике праве обертання

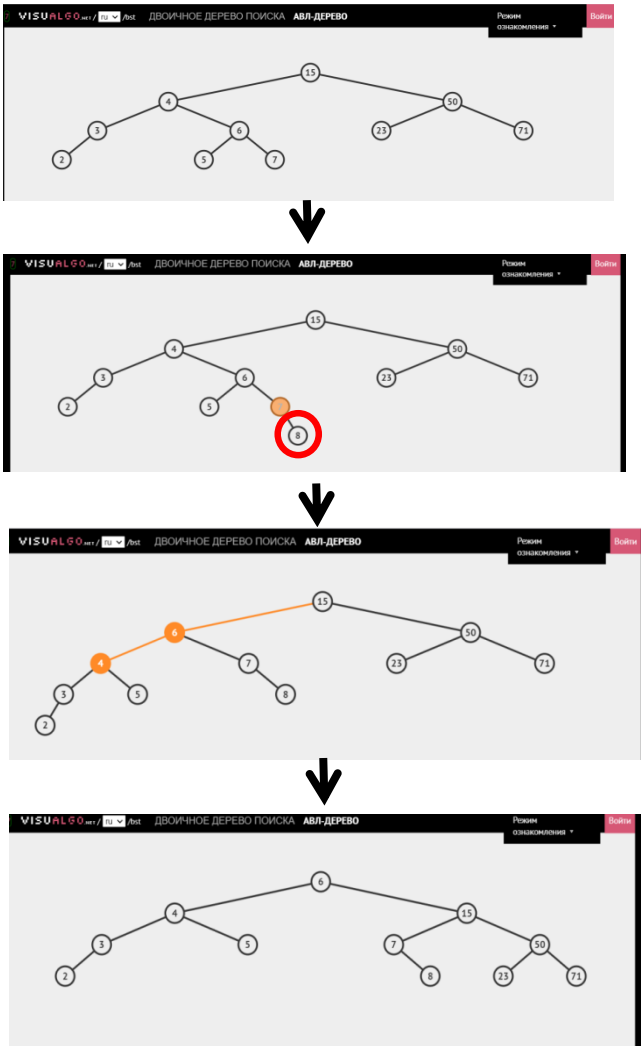

Рис.1.9 Велике праве обертання(виконано у візуалізаторі)

Часова складність

Г.М.Адельсон-Вельський та Е.М.Ландіс довели теорему, згідно якої висота АВЛ-дерева з N внутрішніми вершинами знаходиться між log2(N+1) і 1.4404*log2(N+2) – 0.328, тобто висота АВЛ- дерева ніколи не перевищить висоту ідеально збалансованого дерева більш, ніж на 45%. Для великих N має місце оцінка 1.04*log2(N). Це дозволяє реалізувати швидкий пошук ключа у АВЛ-дереві.

Формалізований опис задачі

Організація АВЛ дерева полягає у реалізації дерева, трьох функцій-операцій над деревом(додавання, пошук, видалення елементів), балансування.

2. Вибір методу розв’язання

Для розв’язання задачі АВЛ-дерево було вирішено реалізувати стандартний метод реалізації АВЛ-дерев, а саме створення 3 основних операцій з АВЛ-деревом: додавання вершини, видалення її та пошук. Потрібно визначити ключові моменти:

1. *Операція додавання вершини*
2. *Операція видалення вершини*
3. *Операція пошуку вершини*

Для реалізації вище зазначеного методу необхідно:

1. Реалізувати пусте АВЛ-дерево
2. Реалізувати операцію додавання вершин до АВЛ-дерева
3. Реалізувати операцію видалення вершин з АВЛ дерева
4. Реалізувати операцію пошуку вершин з АВЛ-дерева
5. Додатково: реалізувати можливість збереження оброблених даних до текстового файлу

Операція додавання вершини

Безпосередньо при вставці новому листу задається нульовий баланс. Процес включення вершини складається з трьох частин:

- Проходу по шляху пошуку, поки не переконаємося, що ключа в дереві немає. Включення нової вершини в дерево і визначення результуючих характеристик
- Повернення назад по шляху пошуку і перевірки характеристики в кожній вершині.
- Якщо необхідно – балансування.

Передбачимо, що процес з лівої гілки повертається до батька (рекурсія йде назад), тоді можливі три випадки:

- висота лівого піддерева < висота правого піддерева: зрівняється висота лівого піддерева = висота правого піддерева. Нічого робити непотрібно.
- висота лівого піддерева = висота правого піддерева: тепер ліве піддерево буде більше на одиницю,але балансування не потрібне.
- висота лівого піддерева > висота правого піддерева: теперь висота лівого піддерева - висота правого піддерева = 2, - потрібне балансування.

У третій ситуації потрібно визначити вид балансування лівого піддерева. Якщо ліве піддерево цієї вершини вище правого, то потрібне велике праве обертання, інакше вистачить малого правого. Аналогічні (симетричні) міркування можна привести і для включення в праве піддерево.

Операція видалення вершини

Процедура видалення //рекурсивна

{Якщо вершина – лист,

{видалити вершину викликати балансування всіх вершин предків видаленої в порядку від батька до кореня}

Інакше {знайти найближчу вершину у піддереві найбільшої висоти (правому або лівому)

переставити її на місце вершини, що видаляється викликати процедуру видалення}}

Даний алгоритм зберігає збалансованість. В результаті вказаних дій процедура видалення викликається не більше 3 разів, оскільки у вершини, що видаляється по другому виклику, немає одного з піддерев. Але пошук найближчого вузла кожного разу вимагає O(N) операцій, для оптимізації пошук найближчої вершини проводиться по краю піддерева. Звідси кількість дій *O(log(N))*. При видаленні,якщо вершина – лист, то вона видаляється, і зворотний обхід дерева походить від батька видаленого листа. Якщо не лист – їй знаходиться «заміна», і зворотний обхід дерева походить від батька «заміни». Безпосередньо після видалення елементу – «заміна» отримує баланс видаленого вузла. При зворотному обході: якщо при переході до батька прийшли зліва – баланс збільшується на 1, якщо ж прийшли справа – зменшується на 1. Це робиться до тих пір, поки при зміні балансу він не стане рівним -1 або 1: в даному випадку така зміна балансу свідчитиме про незмінну різницю висот піддерев. Обертання відбуваються по тих же правилах, що і при вставці.

Операція пошуку вершини

Нехай шукане значення буде item.

- Якщо вершина АВЛ == item: шукане значення корінь дерева, виводимо данні кореня

- Якщо item>кореня, рухаємося вправо, якщо навпаки – вліво, допоки не знайдемо шукане значення. Виведення даних ключа або повідомлення, що ключ у дереві не знайдено.

Вимоги до програмних та технічних засобів

Необхідною вимогою для функціонування програмного продукту є встановлена операційна система класу Windows XP/Vista/7/8 та вище.

3. Алгоритм обраного методу

Нижче на Рис.3.1-3.4 подано укрупнений алгоритм розв’язання задачі.

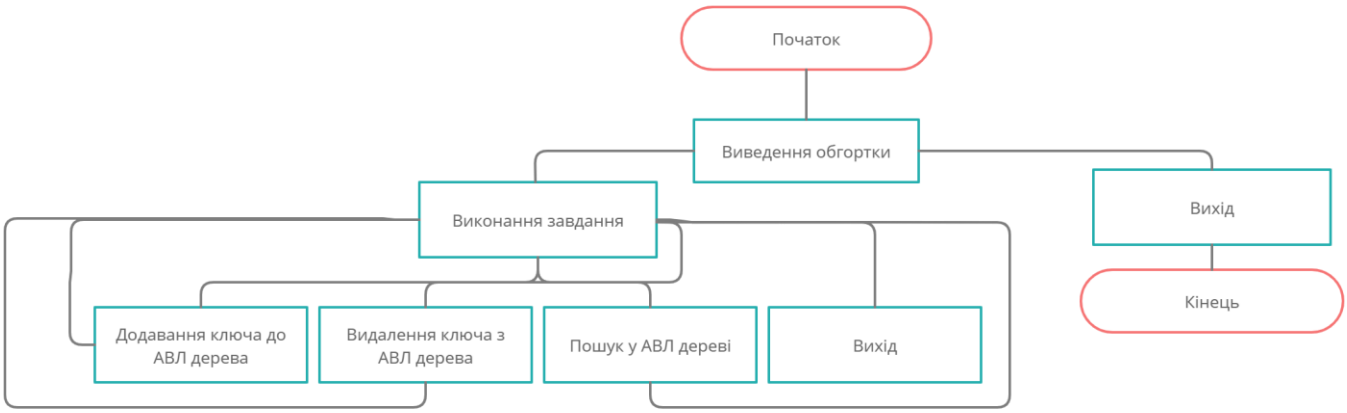

Рис.3.1.Головне меню: вибір підпрограми

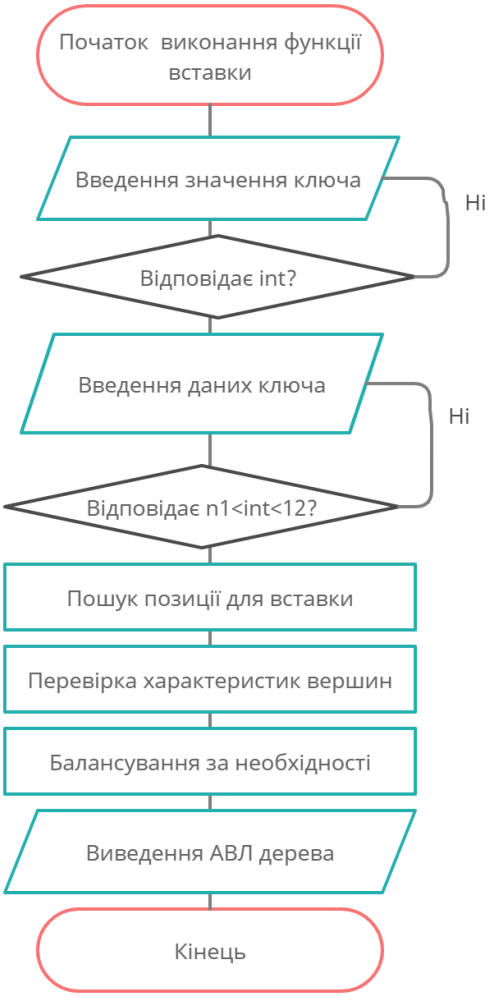

Рис.3.2 Вставка: введення значення нового ключа та даних, пошук місця для ключа, додавання ключа до АВЛ-дерева, балансування

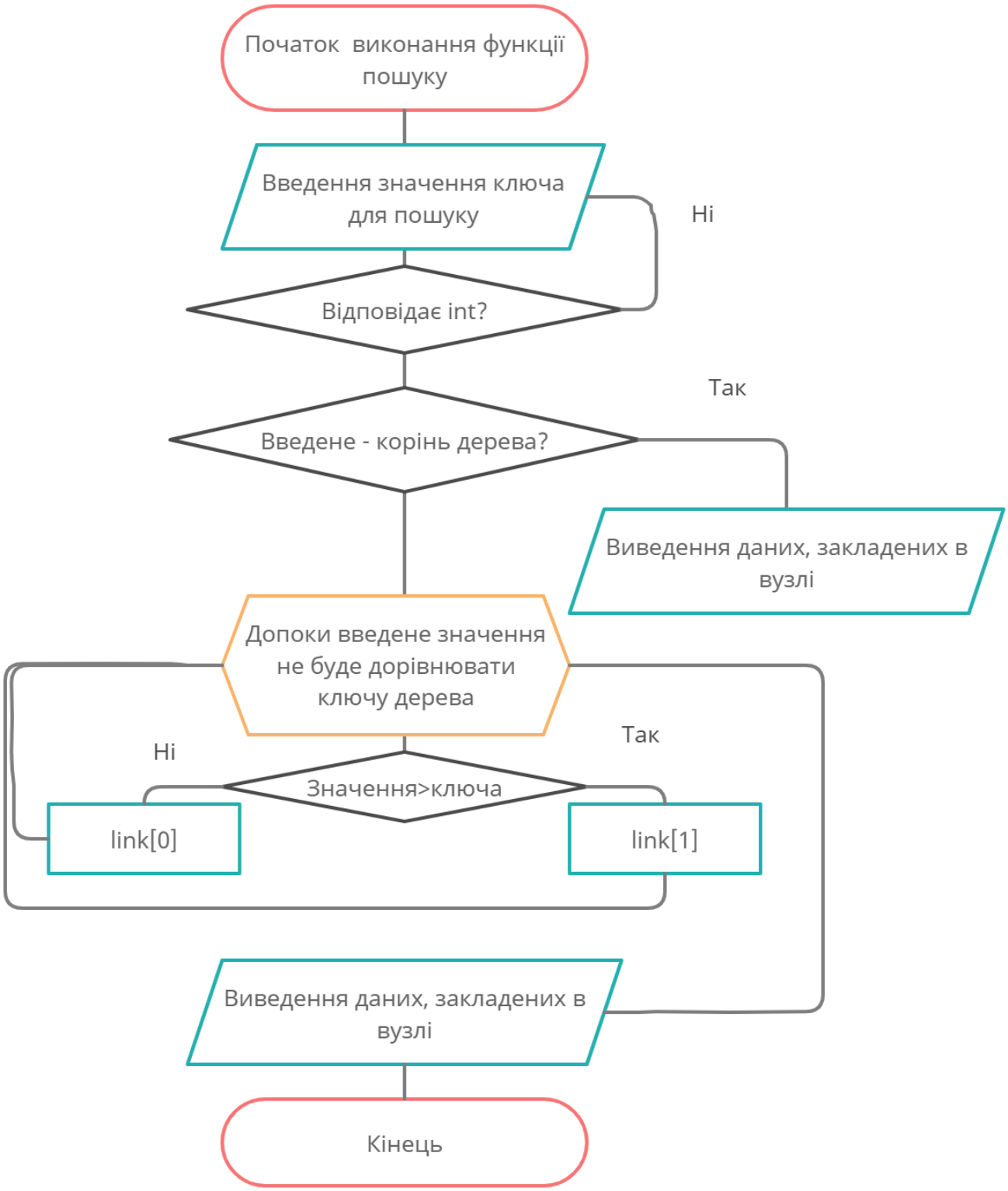

Рис.3.3 Пошук: введення шуканого ключа, пошук його у АВЛ-дереві, виведення значення даних ключа або повідомлення про його відсутність

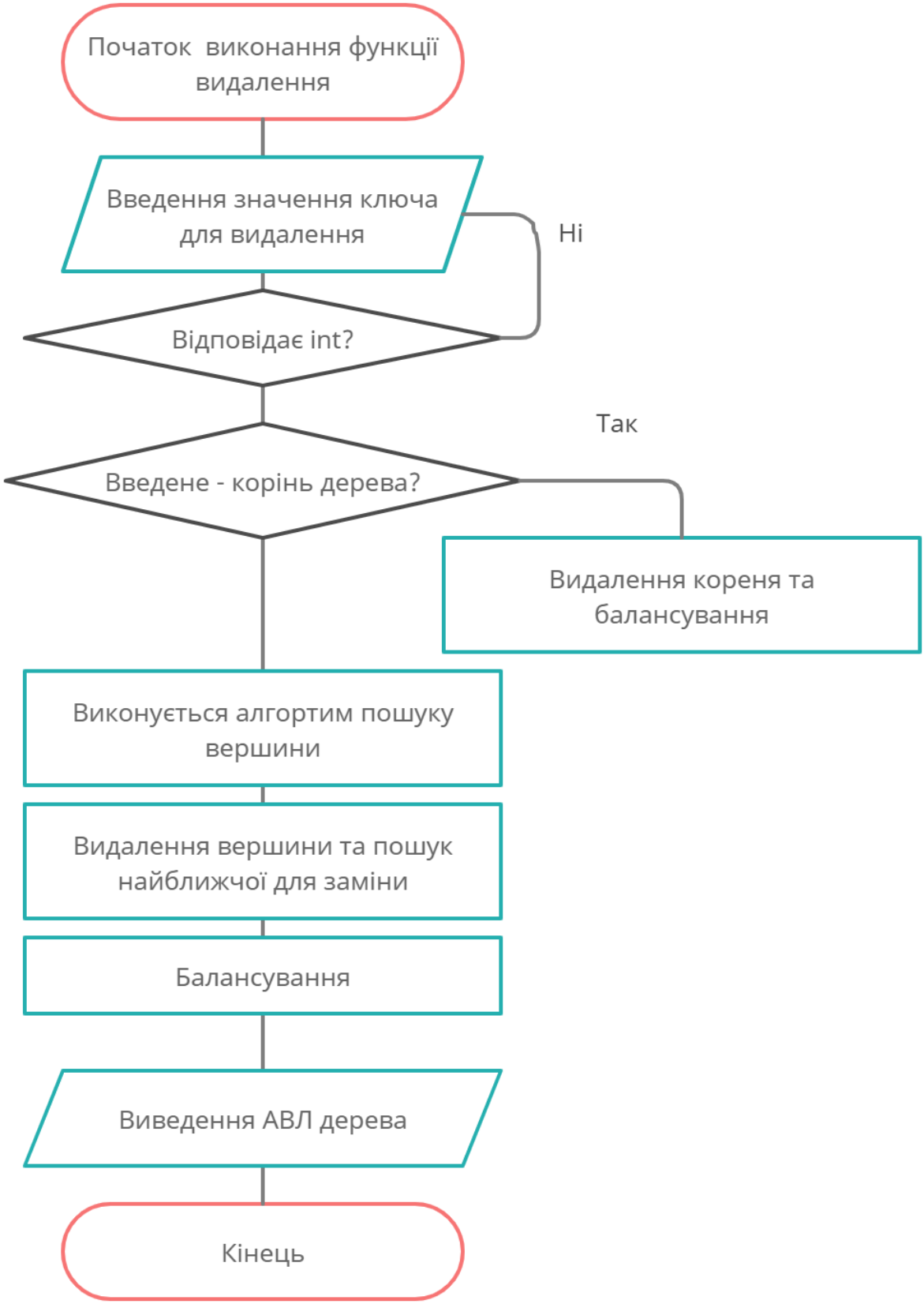

Рис.3.4 Видалення: введення ключа для видалення, пошук його у АВЛ-дереві, його видалення, пошук найближчого “родича”, балансування

4. Опис програми.

Інструкція для кінцевого користувача(інтерфейс)

1. Виведення титульного листа програми

Інтерфейс зображено на Рис.4.1. Зображено обкладинку програми, на якій вказано вид роботи, авторку та її групу, обрану задачу, рік виконання роботи.

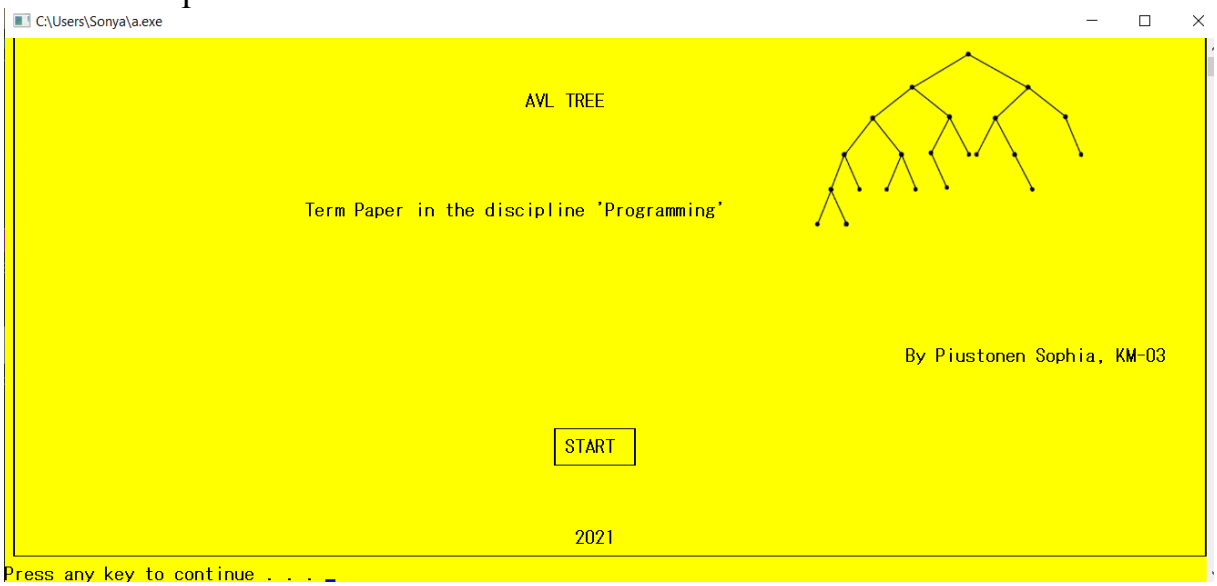

Рис.4.1. Титульний лист програми

Програма реалізована англійською мовою. За допомогою бібліотеки windows.h був змінений колір фону. Змінений шрифт. Поля для введення значень(квадратики) побудовані за допомогою функції GoToXY.

2. Головне меню

Після натиснення любої клавіші для користувача відкривається головне меню. Головне меню складається з двох пунктів: виконання завдання, вихід з програми. Рис.4.2

Рис.4.2.Головне меню програми

3. Меню програм

Користувач починає роботу програми після натиснення ‘1’. Якщо користувач натисне будь-яку іншу клавішу, окрім ‘1/2’, буде надісланий повторний запит на введення. Рис.4.3

Рис.4.3.Меню програм

Після вибору першого пункту меню користувач має можливість обрати один з чотирьох пунктів меню. При виборі ‘1’ користувач запускає підпрограму додавання нового елемента до AVL дерева. Користувач вводить значення нового ключа та значення ключа(знак зодіаку). При введенні будь-якого символу, окрім int, повторний запит на введення значення буде надіслано. Після введення новий вузол додається до AVL-дерева. Повернення до підменю вибору підпрограми. Якщо баланс дерева порушується, то перед виведенням отриманого дерева відбувається балансування. Виведення отриманого дерева. Повернення у меню вибору підпрограм. Виведення АВЛ-дерева виконується за допомогою літер “R”, “L”, що показують напрям руху ключа. Рис.4.4

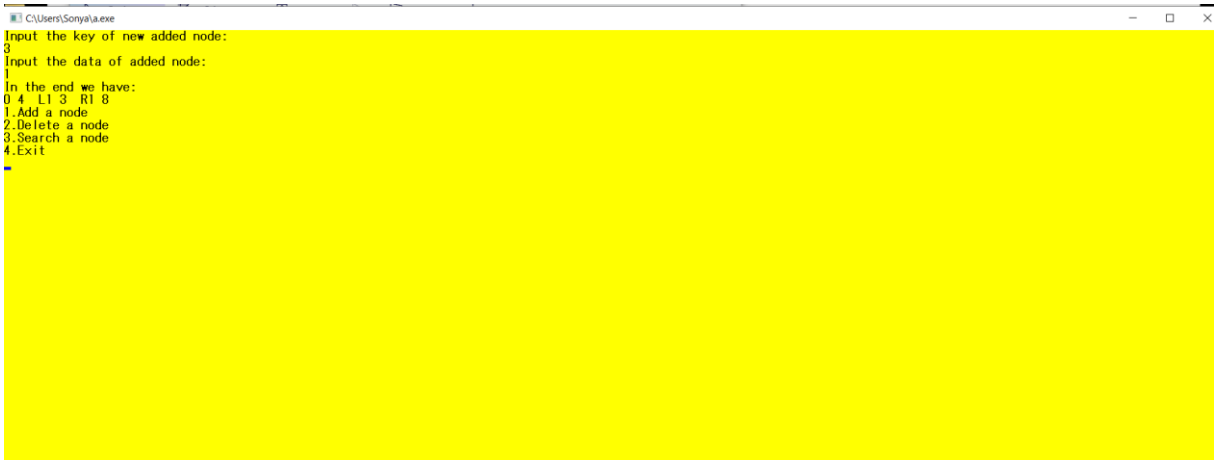

Рис.4.4. Додавання нового елемента до АВЛ-дерева

При виборі ‘2’ користувач запускає підпрограму видалення елемента з AVL дерева. Користувач вводить значення ключа, який необхідно видалити. При введенні будь-якого символу, окрім int, повторний запит на введення значення буде надіслано. Якщо введений ключ є в AVL дереві, то вузол видаляється, а дерево балансується. Якщо такого ключа в AVL дереві немає, завершується процес видалення. Виведення отриманого дерева. Повернення у меню вибору підпрограм.Рис.4.5

Рис.4.5. Видалення уведеного ключа з АВЛ дерева разом із даними

При виборі ‘3’ користувач запускає підпрограму пошуку елемента в AVL дереві. Користувач вводить значення ключа, який необхідно знайти. При введенні будь-якого символу, окрім int, повторний запит на введення значення буде надіслано. Якщо введений ключ є в AVL дереві, виводиться відповідний ключ та його значення(знак зодіаку). Якщо такого ключа в AVL дереві немає, завершується процес пошуку. Виведення отриманого дерева. Повернення у меню вибору підпрограм. Рис.4.6

Рис.4.6. Пошук введеного елемента у АВЛ-дереві та виведення даних веденого ключа

4. Завершення роботи та збереження даних

При виборі ‘4’ користувач повертається у головне меню, а оброблене користувачем дерево зберігається у допоміжний файл “result.txt”. Виведення АВЛ-дерева виконується за допомогою літер “R”, “L”, що показують напрямки знаходження ключів. Рис.4.7.

Рис.4.7. Скріншот збережених у текстовий файл даних

При виборі ‘2’ у головному меню користувач завершує роботу програми.

Інструкція для програміста

Для реалізації АВЛ-дерев використовують спискову пам'ять. Вузол дерева містить значення ключа, поле даних, вказівники на правого і лівого нащадків і характеристику ch, яка дорівнює різниці висот лівого і правого піддерев. Для AVL дерева характеристика (баланс) ∈ {-1,0,1}. Як тільки характеристика виходить за допустиму множину дерево перестає бути АВЛ деревом і потребує балансування. На рисунку 4.8. зображено структуру АВЛ-дерева.

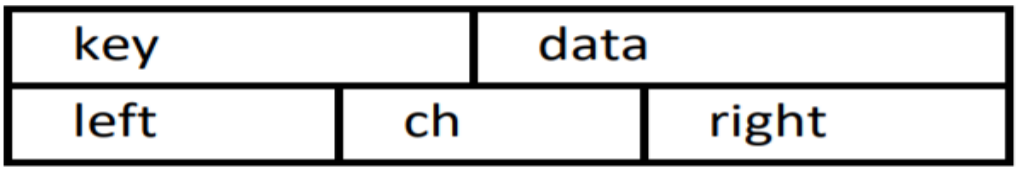

Рис.4.8. Структура АВЛ-дерева

Для роботи програми були імпортовані наступні бібліотеки: stdlib.h, windows.h, ctype.h, stdbool.h, string.h, stdlib.h. Зміна кольору інтерфейсу та реалізація функції GoToXY. Бібліотека *windows.h* працює лише у операційній системі Windows. Функція *system("color XX")* використана для зміни кольору інтерфейсу.

Текстовий опис алгоритму дій програми:

1. За допомогою функції GoToXY виконується організація титульної сторінки: startim
2. Функція main() виконує запуск головного меню програми
3. Після вибору пункту головного меню(avl/exit) “avl” починає свою роботу функція first: меню програм. Меню програм дозволяє обрати один із варіантів виконання задачі:
   3.1. Додавання нової вершини: avl_insert
   3.2. Видалення вершини: avl_delete
   3.3. Пошук вершини
4. При натисканні “4” у меню програм відбувається повернення користувача у головне меню та збереження оброблених даних

Для реалізації програми було використано різноманітні стандартні функції мови С, а також створено декілька допоміжних функцій, детальна

інформація про які наведено у табличці нижче, а також структуру avl_node:

```
      struct avl_node
{
   struct avl_node * link[2];
   int key;
   short ch;
   int let_it_be_your_zodiac;
};
```

Нижче на Рис.4.9 подано схему взаємодії функціональних елементів.

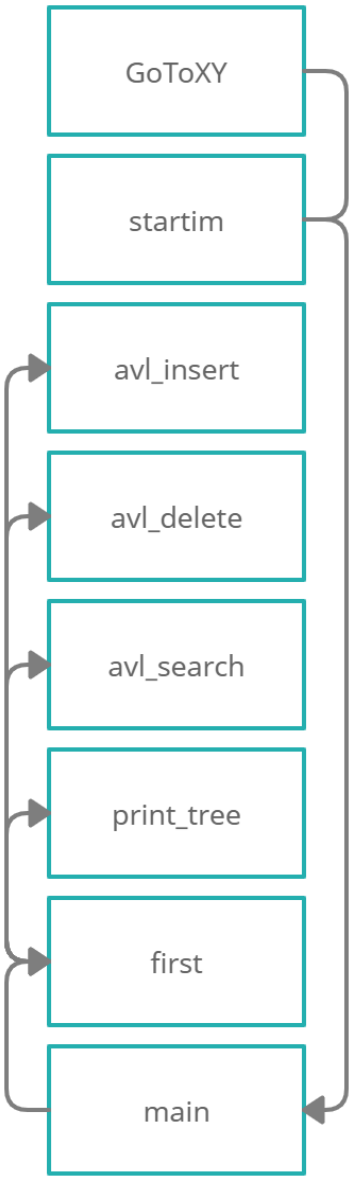

Рис. 4.9 Схема взаємодії функціональних елементів

Назва функції	*Параметри*	*Призначення(результат роботи)*
IntInput	Текст запиту	Перевірка значення на тип int
avl_tree	---	Вказівник на структуру AVL
new_node	Struct avl_tree * tree, int item(ключ), int zodiac(данні)	Новий вузол
avl_insert	Struct avl_tree * tree, int item(ключ), int zodiac(данні)	Додавання вузла до AVL дерева
avl_delete	Struct avl_tree * tree, int item(ключ)	Видалення вузла з AVL дерева.
avl_search	Struct avl_tree * root, int data(ключ)	Пошук введеного елемента, виведення його та даних цього ключа
print_tree	Struct avl_tree * root, int k	Друк дерева
GoToXY	Int column, int line	Зміна координат
startim	---	Виведення обгортки
first	---	Організація меню для вибору підпрограм
main	---	Організація головного меню

Таблиця 4.1. Опис основних функцій

Назва об'єкта	***Опис***	***Призначення***
Link[2]	Тип struct avl_node	Зберігає вказівники на ліве та праве піддерева
key	Тип int	Зберігає значення

		ключа АВЛ дерева
Ch	Тип short	Зберігає значення коефіцієнта балансу
let_it_be_your_zodiac	Тип int	Зберігає данні ключа(знак зодіаку)
Avl_node	Тип struct	Структура АВЛ дерева
*tempw, *tempx, *tempy. *tempv	Тип avl_node	Зберігає вказівники на вузли
Dir	Тип int	Визначає рух по АВЛ дереву, виходячи з отриманого значення. 1-вправо, 0-вліво.
Adummy	Тип int	Допоміжна структура
Flag	Тип int	Слугує для визначення пункту меню
Item	Тип int	Слугує для заповнення значення ключа АВЛ дерева
let_it_be_your_zodiac (main)	Тип int	Слугує для заповнення даних ключа АВЛ дерева
Choice	Тип int	Слугує для вибору пункту головного меню

Таблиця 4.2. Опис основних даних

Int_Input	new_node	avl_insert	avl_delete
У параметр функції переданий тип char – текст, який буде виводити	Функція створення нового вузла. Параметри функції – структура	Функція вставки нового вузла. Виконується за алгоритмом вставки. Введення значення ключа та	Функція видалення вузла відбувається за алгоритмом видалення вершини. Намагання видалити інформацію з пустого

програма за кожного неправильного введення значення. Введення значення error = scanf("%d%c", &res, &term) буде повторюватися, допоки значення типу int не буде введено	дерева, ключ та знак зодіаку у дереві. Динамічно виділяється пам'ять для нового вузла. Усім структурним значенням дерева надаються відповідні значення. Ключу та зодіаку відповідають введені значення, вказівникам – значення NULL(допоки не додані ще вузли до данного)	зодіаку для вставки. Якщо дерево пусте – ми додаємо кореневий елемент. Якщо дерево має>1елемента – відбувається пошук необхідного місця для вставки. Якщо доданий елемент більший за кореневий – відбувається ‘рух’ за правим вказівником(link[1]), інакше – за лівим(link[0]). Вибір 1/0 здійснюється за допомогою змінної dir. Знаходження місця для елемента та його вставка. Зміна всіх коефіцієнтів балансу пройдених елементів. Перевірка дерева на баланс. Якщо баланс порушено, відбувається один з описаних раніше видів балансування. Якщо дерево збалансоване – роботи функції вставки завершено.	дерева(дерево без вузлів) завершиться закінченням роботи програми. Введення значення ключа для видалення. За принципом пошуку місця вершини з функції вставки відбувається пошук вершини для видалення. Відбувається видалення обраного вузла та пошук його замінника(найближчий за значенням ‘родич’). Перенесення значення нового вузла до видаленого. Заміна коефіцієнтів пройдених елементів. Перевірка балансу дерева. Якщо баланс порушено, відбувається один з описаних раніше видів балансування. Якщо дерево збалансоване – роботи функції видалення завершено.

Таблиця 4.3.1. Детальний опис функцій

avl_search	startim	first
У параметр функції передані сформоване дерево та шукане значення. Якщо дерево пусте(без вузлів), робта програми пошуку вершини завершена. Функція організована рекурсивно: допоки значення не буде знайдено, вказівник буде рухатися вправо/вліво(link[0/1]), залежно від значення. Якщо шукане значення не було знайдене – видається відповідне повідомлення. Якщо шукане значення було знайдено – користувач отримує значення даних шуканого ключа – знак зодіаку.	Функція виведення початкової обгортки програми. system("cls") – очистка екрану. Макет обгортки намальваний за допомгою Циклічнх повторень виведення ASCII символів(рамки). За допомгою функція GoToXy реалізовано зміну положень курсору. Обгортка містить ФІО студентки, номер групи, назву теми, рік виконання.	Функція, яка реалізує меню вибору підпрограм роботи з AVL деревом. Всі функції роботи з деревом упорядковані у зручне меню.

Таблиця 4.3.2. Детальний опис функцій

5. Результати

Демонстрацію результатів виконаної роботи проведемо на контрольному прикладі. В якості такого прикладу візьмемо задачу[2].

Завдання: організувати роботу АВЛ-дерева та його головних операцій.

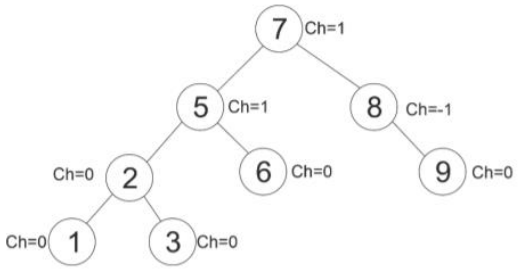

Рис.5.1 АВЛ-дерево(приклад)

Нотуємо значення вершин з контрольного прикладу та вводимо дані до програми. Введені вершини: 1, 2,3,5,6,7,8,9. Рис.5.1.

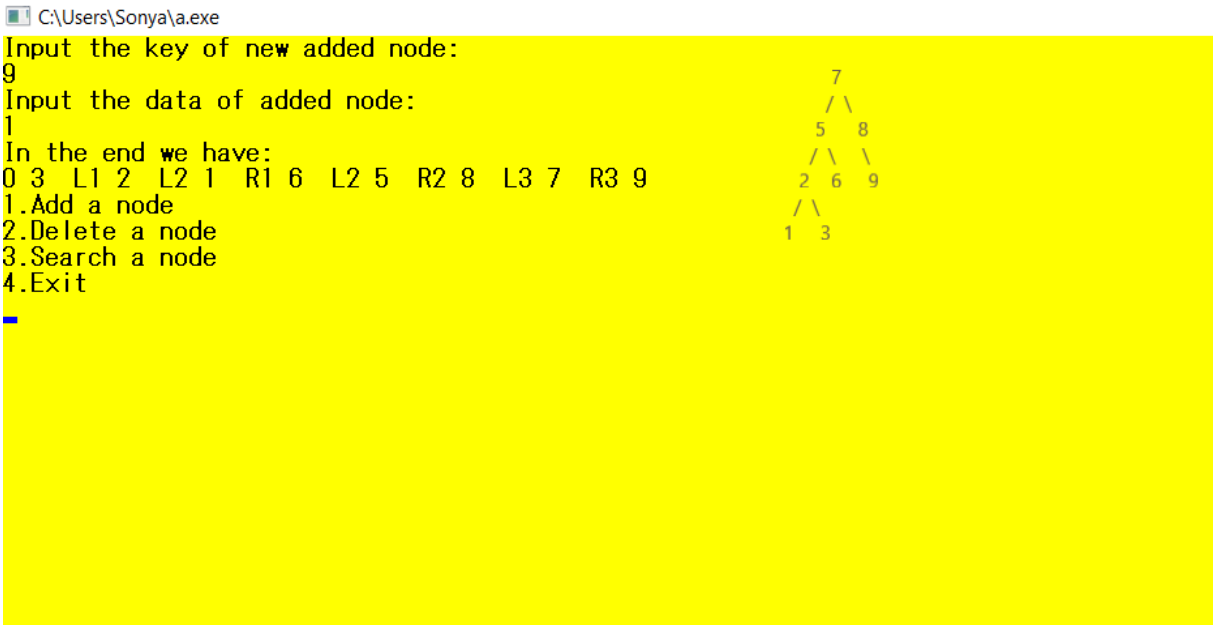

Рис.5.2 Приклад АВЛ-дерева з веденими даними(за контрольним прикладом)

Бачимо, що реалізація додавання вершин відповідає введеним вершинам контрольного прикладу.Рис.5.2.

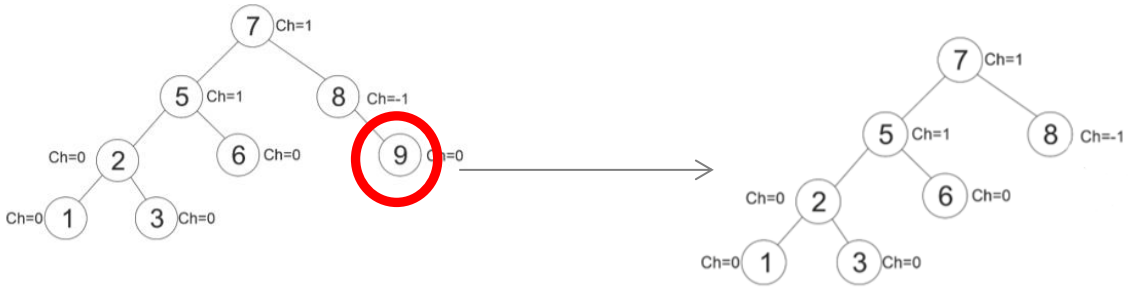

Рис.5.3 Видалення вузла з АВЛ-дерева

У контрольному прикладі подано варіант видалення вершини з ключем “9” та приклад видалення вершини “2” з балансуванням. Зробимо теж саме й у реалізованому програмному продукті. Рис.5.4. Рис.5.6.

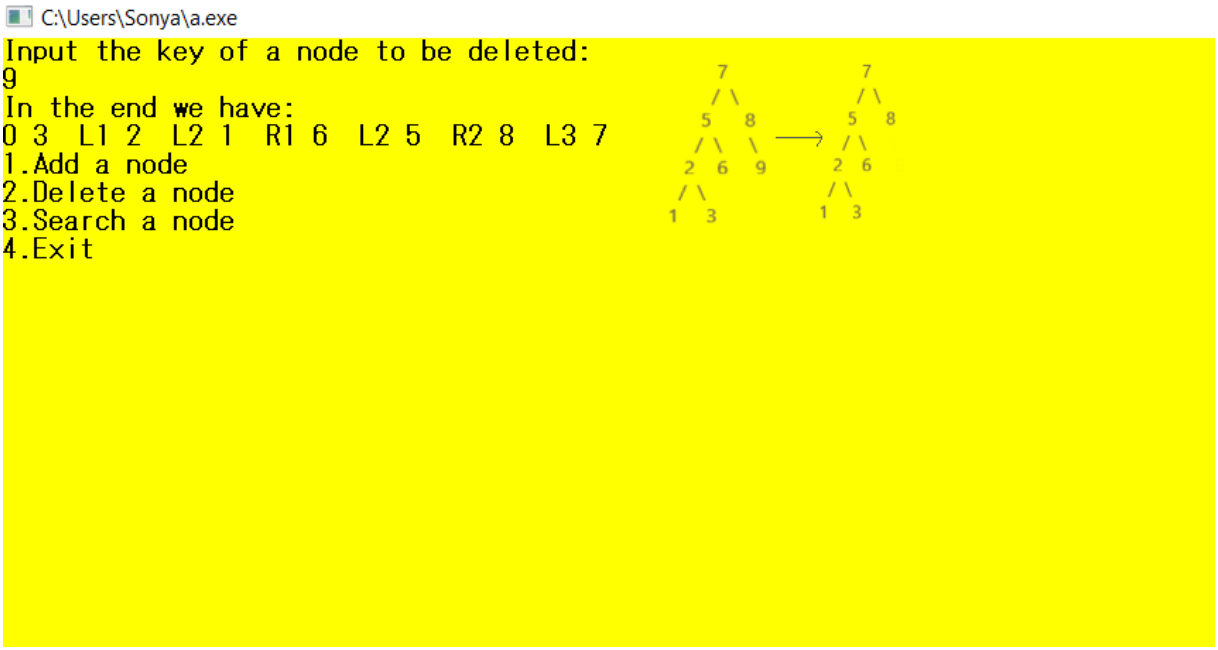

Рис.5.4 Приклад реалізації видалення вузла з АВЛ-дерева

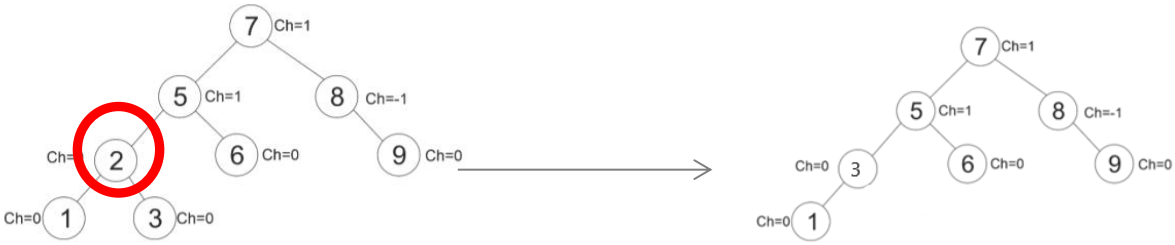

Рис.5.5.Видалення вузла з балансуванням та пошуком “найближчого родича” для заміни

```
C:\Users\Sonya\a.exe
Input the key of a node to be deleted:
2
In the end we have:
0 5  L1 3  L2 1  R1 7  L2 6  R2 8  R3 9
1.Add a node
2.Delete a node
3.Search a node
4.Exit
```

Рис.5.6.Приклад видалення елемента з балансуванням та пошуком “найближчого родича” для заміни

Бачимо, що реалізація видалення вершини відповідає контрольному прикладу.

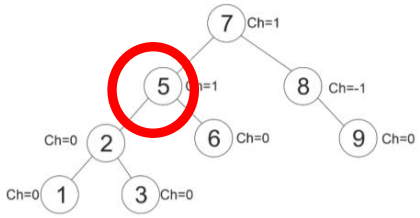

Рис.5.7 Пошук вузла в АВЛ-дереві

У контрольному прикладі реалізовано пошук вершини з ключем “5”. Знайдемо таку ж вершину і у створеному АВЛ-дереві. Рис.5.8.

```
C:\Users\Sonya\a.exe
Input the key of node to search:
5
The data(zodiac): 1
In the end we have:
0 3  L1 2  L2 1  R1 6  L2 5  R2 8  L3 7
1.Add a node
2.Delete a node
3.Search a node
4.Exit
```

Рис.5.8 Приклад реалізації пошуку вузла в АВЛ-дереві

Відбувається збереження ведених даних у файл “result.txt”. Виведення АВЛ-дерева виконується за допомогою літер “R”, “L”, що показують напрямки знаходження ключів. Рис.5.9.

Рис.5.9 Збереження оброблених даних у текстовий файл

Можемо зробити висновок, що отриманий внаслідок виконання програми результат для даних із контрольного прикладу еквівалентний тому, який було подано у книзі. Таким чином, на контрольних прикладах було продемонстровано, що програма правильно виконує операції створення АВЛ-дерева, його балансування, пошуку елементів, видалення та додавання.

ВИСНОВКИ

AVl дерево - це дуже зручний та швидкий метод організації даних, у разі використання якого можна легко знайти будь-які конкретні дані чи виявити, що їх немає. У цьому я переконалась під час

У ході виконання курсової роботи було засвоєно теоретичний матеріал за темою “AVL” та набуття початкових практичних навичок для роботи над задачами, які стосуються алгоритмів AVL. Створено програму, що утворює АВЛ-дерево відповідно до забаганок користувача. Удосконалено вміння роботи з динамічною пам’яттю і структурами даних, утворених користувачем. Інтерфейс програми дружній до користувача, що дозволяє з комфортом виконувати потрібні обчислення. Програма може використовуватись студентами для полегшення навчального процесу компаніями для швидкого пошуку впорядкованої інформації.

Було реалізовано програмний продукт згідно з формалізованим описом задачі. Отже, мети курсової роботи було досягнуто.

Переваги	Недоліки
Зручність у використанні	Відсутність графічного інтерфейсу
User-friendly інтерфейс	Обмеженість її можливості використання без відповідних модифікацій
Циклічність	
Складність алгоритму O(log(n))	
Збереження результатів	

Шляхи подальшого покращення

Покращити програму можна додавши більш детальний графічний інтерфейс із зображеннями АВЛ дерева та роботи її функцій.

Література

1. https://uk.wikipedia.org/wiki/%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE#%D0%9E%D0%BF%D0%B5%D1%80%D0%B0%D1%86%D1%96%D1%97_%D0%B7_%D0%90%D0%92%D0%9B-%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%B0%D0%BC%D0%B8
2. Левитин А. В. Алгоритми: вступ в розробку і аналіз. : Пер. з англ. – М. : Видавничий дім «Вільямс», 2006. – 576с.
3. https://visualgo.net/ru/bst
4. https://studfile.net/preview/3760013/

Додаток А. Текст програми мовою С

```
//Підключення необхідних для роботи бібліотек

#include <stdio.h>
#include <stdlib.h>
#include <Windows.h>
#include <ctype.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h>

// Функція для перевірки введеного значення на тип int
int IntInput(char* text) {
    int res = 0;
    int error = 0;
    char term;

    do {
        printf("%s", text);
        error = scanf("%d%c", &res, &term);
        fflush(stdin);
    } while(error != 2 || term != '\n');

    return  res;
}

//Стандартна структура  AVL, яка складається з двох вказівників на ліве та
//праве піддерево, значення ключа, коефіцієнту балансу та значення, яке
//лежить у ключі(знак зодіаку)
struct avl_node
{
```

```
    struct avl_node * link[2];
    int key;
    short ch;
    int let_it_be_your_zodiac;
};

//Структура-вказівник на AVL дерево
struct avl_tree
{
    struct avl_node * root;
};

//Функція, яка створює новий вузол AVL дерева, якому надаються
//значення вказані в функції

struct avl_node * new_node(struct avl_tree * tree, int item, int zodiac)
{
    struct avl_node * node = malloc(sizeof * node);
    node->key = item;
    node->link[0] = node->link[1] = NULL;
    node->ch = 0 ;
    node->let_it_be_your_zodiac=0;
    node->let_it_be_your_zodiac=zodiac;
    return node;
};

//Функція, яка реалізує вставку введеного ключа та даних у AVL дерево
int avl_insert(struct avl_tree * tree, int item, int zodiac)
{
    struct avl_node ** first, *tempw, *tempx, *tempy, *tempv;
    first = &tree -> root;
    tempx = tempv = tree->root;
```

```
    if(tempx == NULL) //випадок, коли ми маємо пусте дерево
    {
        tree->root = new_node(tree,item, zodiac);
        return tree -> root != NULL;
    }


    // пошук позиції для вставки елемента. Елементи більші за корінь знаходяться у правій частині, а менші – у лівій частині
    for(;;)
    {
        int dir;
        // якщо елемент є - закінчуємо
        if(item == tempv->key) return 2;

        dir = (item > tempv->key) ; //якщо умова виконується, то надається значення 1, інакше 0
        tempy = tempv -> link[dir];
        if(tempy == NULL)
        {
            tempy = tempv -> link[dir] = new_node(tree,item, zodiac);
            if(tempy == NULL) return 0;
            break;
        }
        if(tempy->ch!=0)
        {
            first = &tempv -> link[dir];
            tempx = tempy;
        }
        tempv = tempy;
```

```
}

// змінюємо коефіцієнти балансу
tempw = tempv = tempx -> link[item > tempx->key];
while(tempv != tempy)
    if(item < tempv->key)
    {
        tempv -> ch = -1;
        tempv = tempv -> link[0];
    }
    else
    {
        tempv -> ch=   +1;
        tempv = tempv -> link[1];
    }

// баланс для додавання елементів у ліве піддерево
if(item < tempx->key)
{
    if (tempx ->ch != -1)
        tempx -> ch --;
    else if(tempw ->ch == -1) //виконується малий поворот вправо
    {
        *first = tempw;
        tempx -> link[0] = tempw -> link[1];
        tempw -> link[1] = tempx;
        tempx ->ch = tempw -> ch = 0;
    }
```

```
  //виконується великий поворот вправо
    else
    {
        *first = tempv = tempw -> link[1];
        tempw -> link[1] = tempv -> link[0];
        tempv -> link[0] = tempw;
        tempx -> link[0] = tempv -> link[1];
        tempv -> link[1] = tempx;
        if(tempv -> ch == -1)
        {
            tempx -> ch = 1;
            tempw ->ch = 0;
        }
        else if(tempv -> ch== 0)
            tempx -> ch= tempw -> ch = 0;
        else
        {
            tempx -> ch = 0;
            tempw ->ch = -1;
        }
        tempv -> ch=0;
    }
}
    // баланс для додавання елементів у праве піддерево
else
{
    if( tempx -> ch != +1)
        tempx -> ch++;
    else if(tempw -> ch == +1)
    {
```

```
            *first = tempw;
            tempx->link[1] = tempw -> link[0];
            tempw->link[0] = tempx;
            tempx -> ch = tempw ->ch = 0;
        }
        else
        {
            *first = tempv = tempw -> link[0];
            tempw -> link[0] = tempv -> link[1];
            tempv -> link[1] = tempw;
            tempx -> link[1] = tempv-> link[0];
            tempv -> link[0] = tempx;
            if(tempv -> ch == +1)
            {
                tempx -> ch = -1;
                tempw -> ch = 0;
            }
            else if(tempv -> ch == 0)
                tempx -> ch = tempw -> ch = 0;
            else
            {
                tempx -> ch = 0;
                tempw ->ch = 1;
            }
            tempv -> ch = 0;
        }
    }
    return 1;
}
```

```
//Функція, яка реалізує видалення введеного ключа та даних з AVL дерева
int avl_delete(struct avl_tree * tree, int item)
{

    struct avl_node * anode[32];
    int adummy[32];
    int k = 1;
    struct avl_node ** tempy, * tempz;
    adummy[0] = 0 ;
    anode[0] = (struct avl_node * ) &tree -> root;
    tempz = tree -> root;

//пошук введеного елемента та його видалення
    for(;;)
    {
        int dir;
        if(tempz == NULL)
            return 0 ;
        if (item == tempz->key)
            break;

        dir = item > tempz->key;
        anode[k] = tempz;
        adummy[k++] = dir;
        tempz = tempz -> link[dir];
    }

    tempy = &anode[k - 1] -> link[adummy[k-1]];
    if(tempz->link[1] == NULL)
        *tempy = tempz -> link[0];
```

```
else
{
   struct avl_node *tempx = tempz -> link[1];
   if (tempx -> link[0] == NULL)
   {
      tempx->link[0] = tempz -> link[0];
      *tempy = tempx;
      tempx->ch = tempz ->ch;
      adummy[k] = 1;
      anode[k++] = tempx;
   }
   else
   {
      struct avl_node *tempw = tempx -> link[0];
      int j = k++;
      adummy[k] = 0 ;
      anode[k++] = tempx;
      while (tempw -> link[0] != NULL)
      {
         tempx = tempw;
         tempw = tempx -> link[0];
         adummy[k] = 0 ;
         anode[k++] = tempx;
      }

      adummy[j] = 1;
      anode[j] = tempw;
      tempw -> link[0] = tempz -> link[0];
      tempx -> link[0] = tempw -> link[1];
      tempw -> link[1] = tempz -> link[1];
```

```
            tempw -> ch = tempz -> ch;
            *tempy = tempw;
        }
    }
    free(tempz);

// Балансування отриманого дерева
    while(--k)
    {
        struct avl_node *tempw, *tempx;
        tempw = anode[k];
        if (adummy[k] == 0 )
        {
            if (tempw -> ch == -1)
            {
                tempw -> ch = 0;
                continue;
            }
            else if (tempw -> ch == 0 )
            {
                tempw -> ch = 1;
                break;
            }

            tempx = tempw -> link[1];
            if (tempx->ch > -1)
            {
                tempw -> link[1]= tempx -> link[0];
                tempx -> link[0] = tempw;
                anode[k-1] -> link[adummy[k-1]] = tempx;
```

```
    if (tempx -> ch == 0 )
    {
        tempx -> ch = -1;
        break;
    }
    else
        tempw ->ch = tempx ->ch = 0 ;
}
else
{
    tempz = tempx -> link[0];
    tempx -> link[0] = tempz -> link[1];
    tempz -> link[1] = tempx;
    tempw -> link[1] = tempz -> link[0];
    tempz -> link[0] = tempw;
    if (tempz -> ch == 1)
    {
        tempw -> ch = -1;
        tempx ->ch = 0;
    }
    else if (tempz -> ch == 0 )
        tempw -> ch = tempx -> ch = 0;
    else
    {
        tempw -> ch = 0;
        tempx -> ch = 1;
    }
    tempz -> ch = 0;
    anode[k-1]->link[adummy[k-1]] = tempz;
}
```

```
}
else
{
   if (tempw ->ch == 1)
   {
      tempw -> ch= 0 ;
      continue;
   }
   else if (tempw ->ch==0)
   {
      tempw ->ch = -1;
      break;
   }

   tempx = tempw -> link[0];
   if (tempx -> ch < 1)
   {
      tempw -> link[0] = tempx -> link[1];
      tempx -> link[1] = tempw;
      anode[k-1] -> link[adummy[k-1]] = tempx;
      if (tempx ->ch == 0 )
      {
         tempx ->ch = 1;
         break;
      }
      else
         tempw ->ch= tempx -> ch = 0;
   }
   else if (tempx -> ch == 1)
   {
```

```
                tempz = tempx -> link[1];
                tempx -> link[1] = tempz -> link[0];
                tempz -> link[0] = tempx;
                tempw -> link[0] = tempz -> link[1];
                tempz -> link[1] = tempw;
                if (tempz ->ch == -1)
                {
                   tempw -> ch = 1;
                   tempx -> ch = 0;
                }
                else if (tempz -> ch == 0 )
                   tempw -> ch = tempx -> ch = 0 ;
                else
                {
                   tempw -> ch = 0 ;
                   tempx -> ch= -1;
                }
                tempz -> ch = 0;
                anode[k-1] -> link[adummy[k-1]] = tempz;
            }
        }
    }
    return 1;
}

//Функція, яка реалізує пошук введеного ключа та даних в AVL дерева
int avl_search(struct avl_node* root, int data)
{
    if (root==0){
        printf("\nYou can't delete anything from the empty tree");
```

```
        return 0;
    }
    if (root == NULL)
    {
        printf("Needed node not found\n");
        return 0;
    }
    else if ((root->key == data))
    {
        //printf("Found the needed node!\n");
        printf("The data(zodiac): %d\n", root->let_it_be_your_zodiac);
        return 0;
    }
    if (data < root->key)
        avl_search(root -> link[0], data);
    else avl_search(root -> link[1], data);

}

//Функція, яка друкує AVL дерево
void print_tree(struct avl_node* root, int k)
{
    printf("%d %d  ", k, root->key);
    if (root->link[0] != NULL)
    {
        k++;
        printf("L");
        print_tree(root->link[0], k);
    }
```

```
    if (root->link[1] != NULL)
    {
        if(root->link[0] == NULL)
        {
            k++;
        }
        printf("R");
        print_tree(root->link[1], k);
    }
}

// Функція для вибору зміни положення курсору
void GoToXY(int column, int line)
{
    COORD coord;
    coord.X = column;
    coord.Y = line;

    HANDLE hConsole = GetStdHandle(STD_OUTPUT_HANDLE);
    if (!SetConsoleCursorPosition(hConsole, coord))
    {
    }
}
//Функція для виведення обгортки курсової роботи
void startim()
{
    system("cls");
    printf("%c", (char)218);
    for (int i = 0; i < 118; i++)
    {
```

```
    printf("%c", (char)196);
}
printf("%c", (char)191);
for (int j = 0; j < 28; j++)
{
    printf("%c", (char)179);
    for (int k = 0; k < 15; k++)
    {
        printf("\t");
    }
    printf("%c", (char)179);
}
printf("%c", (char)192);
for (int p = 0; p < 118; p++)
{
    printf("%c", (char)196);
}
printf("%c\n", (char)217);

GoToXY(54, 22);
printf("%c", (char)218);
for (int l = 0; l < 7; l++)
{
    printf("%c", (char)196);
}
printf("%c", (char)191);

GoToXY(54, 23);
printf("%c", (char)179);
printf(" START ");
```

```
    printf("%c", (char)179);

    GoToXY(54, 24);
    printf("%c", (char)192);
    for (int l = 0; l < 7; l++)
    {
        printf("%c", (char)196);
    }
    printf("%c\n", (char)217);

    GoToXY(52, 4);
    printf("AVL TREE");

    GoToXY(32, 10);
    printf("Term Paper in the discipline 'Programming'");

    GoToXY(90, 18);
    printf("By Piustonen Sophia, KM-03");

    GoToXY(57, 28);
    printf("2021\n\n");

    system("pause");
}

//Функція для визову функцій для роботи з АВЛ деревом(за вибором користувача)
int first(){
    int flag, m_flag = 0, item, let_it_be_your_zodiac;
    system("cls");
```

```
printf("\nWhat do you want to do with a given tree? \n");
printf("1.Add a node\n2.Delete a node\n3.Search a node\n4.Exit\n");
GoToXY(20, 15);
printf("%c", (char)218);
for (int l = 0; l < 3; l++)
{
    printf("%c", (char)196);
}
printf("%c", (char)191);

GoToXY(20, 16);
printf("%c", (char)179);
printf("   ");
printf("%c", (char)179);

GoToXY(20, 17);
printf("%c", (char)192);
for (int l = 0; l < 3; l++)
{
    printf("%c", (char)196);
}
printf("%c\n", (char)217);
GoToXY(22, 16);
flag = IntInput("");
system("cls");
while (flag > 4 || flag < 1)
{
    GoToXY(20, 18);
    printf("Incorrect data, try again:\n");
    GoToXY(22, 16);
```

```
        flag = IntInput("");
    }
    struct avl_tree tree;
    tree.root = NULL;
    system("cls");
    printf("\nAvl doesn't contain the same keys:)\n");
    while (flag != 4) {
        if (flag == 1) {
            item = IntInput("Input the key of new added node: \n");
            let_it_be_your_zodiac = IntInput("Input the data (zodiac) of added node:
\n");
            while (let_it_be_your_zodiac>12 ||
let_it_be_your_zodiac<1){let_it_be_your_zodiac = IntInput("Input the data
(zodiac) of added node: \n");}
            avl_insert(&tree, item, let_it_be_your_zodiac);
            printf("In the end we have:\n");
            print_tree(tree.root, 0);
            printf("\nWhat do you want to do with a given tree? \n");
            printf("1.Add a node\n2.Delete a node\n3.Search a node\n4.Exit\n");
            GoToXY(20, 15);
            printf("%c", (char)218);
            for (int l = 0; l < 3; l++)
            {
                printf("%c", (char)196);
            }
            printf("%c", (char)191);

            GoToXY(20, 16);
            printf("%c", (char)179);
            printf("   ");
```

```
    printf("%c", (char)179);

    GoToXY(20, 17);
    printf("%c", (char)192);
    for (int l = 0; l < 3; l++)
    {
        printf("%c", (char)196);
    }
    printf("%c\n", (char)217);
    GoToXY(22, 16);
    flag = IntInput("");
    system("cls");
    while (flag > 4 || flag < 1)
    {
        GoToXY(20, 18);
        printf("Incorrect data, try again:\n");
        GoToXY(22, 16);
        flag = IntInput("");
    }
    system("cls");
}

else if(flag == 2) {
    if (tree.root!=NULL){
    item = IntInput("Input the key of a node to be deleted: \n");
    avl_delete(&tree, item);
    printf("In the end we have:\n");
    print_tree(tree.root, 0);}
    else{printf("\nYou can't delete anything from the empty tree");}
    printf("\nWhat do you want to do with a given tree? \n");
```

```
printf("1.Add a node\n2.Delete a node\n3.Search a node\n4.Exit\n");
GoToXY(20, 15);
printf("%c", (char)218);
for (int l = 0; l < 3; l++)
{
   printf("%c", (char)196);
}
printf("%c", (char)191);

GoToXY(20, 16);
printf("%c", (char)179);
printf("   ");
printf("%c", (char)179);

GoToXY(20, 17);
printf("%c", (char)192);
for (int l = 0; l < 3; l++)
{
   printf("%c", (char)196);
}
printf("%c\n", (char)217);
GoToXY(22, 16);
flag = IntInput("");
system("cls");
while (flag > 4 || flag < 1)
{
   GoToXY(20, 18);
   printf("Incorrect data, try again:\n");
   GoToXY(22, 16);
   flag = IntInput("");
```

```
    }
    system("cls");
}

else if (flag == 3) {
    if(tree.root!=NULL){
    item = IntInput("Input the key of node to search: \n");
    avl_search(tree.root, item);
    printf("In the end we have:\n");
    print_tree(tree.root, 0);}
    else{printf("\nYou can't search anything from the empty tree");}
    printf("\nWhat do you want to do with a given tree? \n");
    printf("1.Add a node\n2.Delete a node\n3.Search a node\n4.Exit\n");
    GoToXY(20, 15);
    printf("%c", (char)218);
    for (int l = 0; l < 3; l++)
    {
        printf("%c", (char)196);
    }
    printf("%c", (char)191);

    GoToXY(20, 16);
    printf("%c", (char)179);
    printf("   ");
    printf("%c", (char)179);

    GoToXY(20, 17);
    printf("%c", (char)192);
    for (int l = 0; l < 3; l++)
    {
```

```
            printf("%c", (char)196);
        }
        printf("%c\n", (char)217);
        GoToXY(22, 16);
        flag = IntInput("");
        system("cls");
        while (flag > 4 || flag < 1)
        {
            GoToXY(20, 18);
            printf("Incorrect data, try again:\n");
            GoToXY(22, 16);
            flag = IntInput("");
        }
        system("cls");
    }
 FILE *result; //додавання оброблених даних до файлу
result = fopen("result.txt", "w");
fprintf(result, "In the end we have:\n");
output_tree(tree.root, 0, result);
fclose(result);


  }

};

//Функція для вибору пункту програми(виконання завдання/вихід)
int main(void)
{
```

```
    system("color E0");
    startim();
    system("cls");
    HANDLE out_handle = GetStdHandle(STD_OUTPUT_HANDLE);
    COORD maxWindow = GetLargestConsoleWindowSize(out_handle);   //the
max size of the window
    SMALL_RECT srctWindow = { 0, 0, maxWindow.X - 1, maxWindow.Y - 1
};
    SMALL_RECT minWindow = { 0, 0, 0, 0 };
    SetConsoleWindowInfo(out_handle, true, &minWindow);
    SetConsoleScreenBufferSize(out_handle, maxWindow);
    SetConsoleWindowInfo(out_handle, true, &srctWindow);

    char* name = calloc(100, sizeof(char));
    GoToXY(80, 60);
    int choice = 1;
    while (true)
    {
        system("cls");
        printf("\n\n\t\tChoose the option:\n\t\t1. AVL.\n\t\t2. Exit.\n");
        GoToXY(20, 15);
        printf("%c", (char)218);
        for (int l = 0; l < 3; l++)
        {
            printf("%c", (char)196);
        }
        printf("%c", (char)191);

        GoToXY(20, 16);
        printf("%c", (char)179);
```

```
printf("   ");
printf("%c", (char)179);

GoToXY(20, 17);
printf("%c", (char)192);
for (int l = 0; l < 3; l++)
{
   printf("%c", (char)196);
}
printf("%c\n", (char)217);
GoToXY(22, 16);
choice = IntInput("");
while (choice > 9 || choice < 0)
{
   GoToXY(20, 18);
   printf("Incorrect data, try again:\n");
   GoToXY(22, 16);
   choice = IntInput("");
}
system("cls");
switch (choice) {
   case 1:
   {
      first();
      int choice_1;
      choice_1 = IntInput("Do you want to return to the menu(1): \n");
      if (choice_1==1){
         break;
      }
```

```
            }
            case 2:
            {
                return 0;
            }

            case 0:
            {
                exit(1);
                break;
            }
        }
        system("cls");
    }
    free(name);
    return 0;
}
```