Міністерство освіти і науки України
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

Кафедра програмування

ЛАБОРАТОРНА РОБОТА № 11

AVL дерева

з курсу “Алгоритми та структури даних”

Виконала:

Студентка групи ПМІ-13

Демко Сніжана Іванівна

Львів – 2024

Мета: ознайомлення з структурою AVL-дерев, реалізація основних операцій з ними, таких як вставка, видалення та пошук, а також тестування цих операцій для перевірки їх правильності.

Поняття **дерева (tree)** виводять з теорії графів, але воно отримало широке застосування в інформатиці. Тому в літературі трапляється два визначення дерева:

1) на основі теорії графів: дерево - це зв'язний ациклічний граф;

2) рекурсивне: дерево - це скінчена множина T, яка складається з одного чи більше вузлів таких, що

- є один спеціальний вузол, який називають коренем даного дерева;
- решта вузлів входять до складу m>0 множин T1,...,Tm, які попарно не перетинаються і кожна з яких сама є деревом (T1,--Tm, називають піддеревами).

Існують **двійково-пошукові (двійкові)** дерева. Ці дерева визначають так: з кожного вузла виходить не більше двох гілок, і кожна гілка ідентифікується як ліва або як права.

AVL-дерево — збалансоване по висоті двійкове дерево пошуку: для кожної його вершини висота її двох піддерев відрізняється не більше ніж на одиницю. AVL-дерево ніколи не буде більш ніж на 45% вище від відповідного ідеального збалансованого дерева, незалежно від кількості вузлів.

Зі збалансованими деревами можна виконувати наступні операції за 0(log2n) одиниць часу, навіть у найгіршому випадку:

- знайти вузол із заданим значенням;
- включити вузол із заданим значенням;
- видалити вузол із заданим значенням.

Операції з AVL-деревом:

- **Вставка:** При вставці нового вузла до AVL-дерева, необхідно спочатку виконати стандартну вставку, а потім перевірити баланс кожного вузла по шляху від нового вузла до кореня. Якщо баланс будь-якого вузла стає більшим за 1 або меншим за -1, то потрібно виконати операції обертання дерева, щоб відновити баланс.
- **Видалення:** При видаленні вузла з AVL-дерева, також необхідно спочатку виконати стандартне видалення, а потім перевірити баланс кожного вузла по шляху від видаленого вузла до кореня. Якщо баланс будь-якого вузла стає більшим за 1 або меншим за -1, то потрібно виконати операції обертання дерева, щоб відновити баланс.

- **Пошук:** Операція пошуку в AVL-дереві виконується так само, як і в звичайному бінарному дереві пошуку. Починаючи з кореня, порівнюємо шуканий ключ з ключем поточного вузла і рухаємося вліво або вправо, поки не знайдемо вузол або не дійдемо до кінця дерева.

Властивості AVL-дерева:

- **Баланс:** Для кожного вузла різниця між висотами лівого та правого піддерева (баланс) має бути -1, 0 або 1.
- **Швидкість операцій:** Основні операції (вставка, видалення, пошук) в AVL-дереві мають логарифмічну складність в середньому, оскільки дерево завжди збалансоване.
- **Пам'ять:** AVL-дерева потребують більше пам'яті для зберігання даних порівняно з іншими бінарними деревами через додаткову інформацію про баланс.

Застосування AVL-дерев:

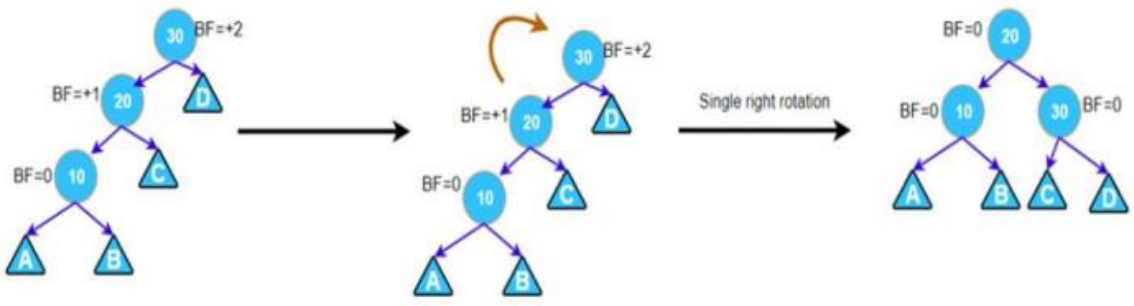
- AVL-дерева зазвичай використовуються там, де потрібно забезпечити ефективні операції пошуку, вставки та видалення впорядкованих даних, наприклад, в базах даних, компіляторах, операційних системах тощо.
- Інші застосування включають використання AVL-дерев для підтримки інтервальних дерев, дерев мережевих маршрутів, дерев фрагментації диска та інших.

Обертання AVL

Щоб збалансувати дерево AVL, при вставці або видаленні вузла з дерева виконуються обертання.

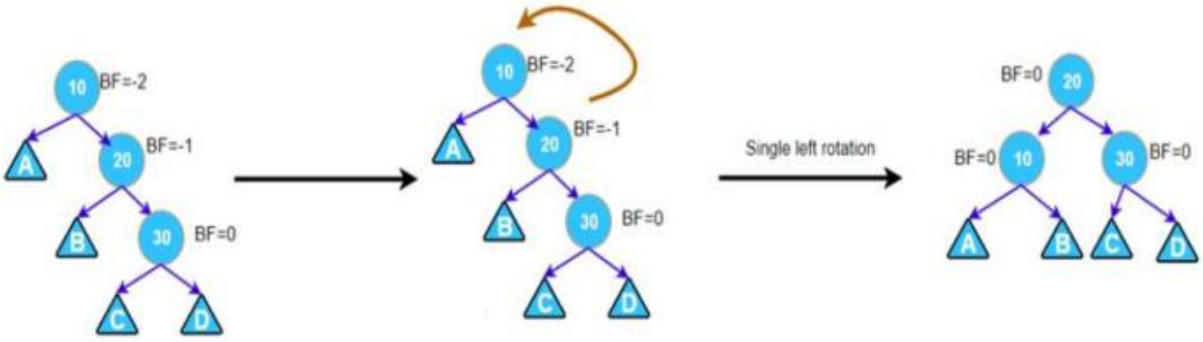
LL - Ліве обертання (Left-Left Rotation): Це обертання виконується, коли незбалансованість виникає через зростання висоти лівого піддерева лівого нащадка кореня. Ось як це працює:

- Корінь лівого піддерева стає новим коренем.
- Піддерево правого сина кореня стає лівим піддеревом попереднього кореня.
- Корінь дерева стає правим сином нового кореня.



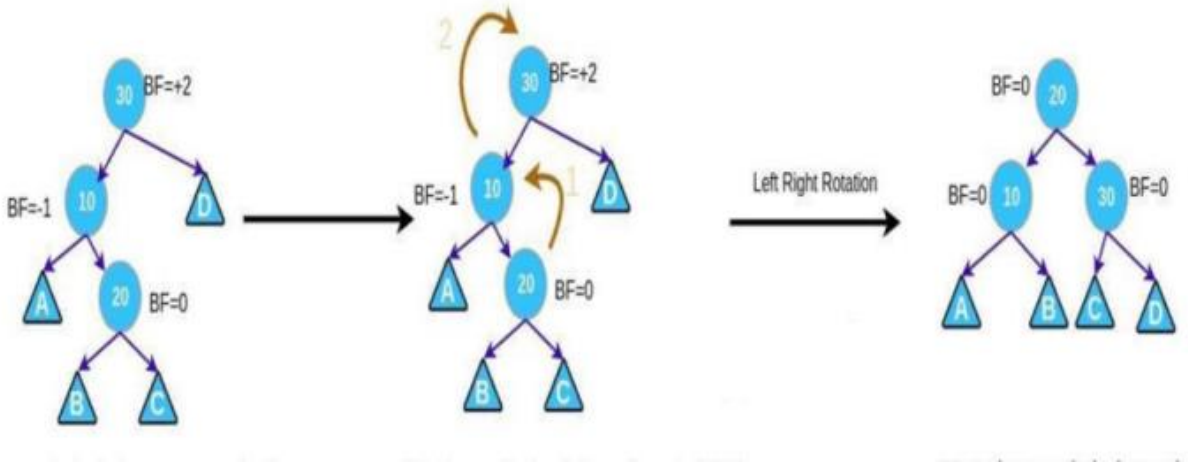
RR - Праве обертання (Right-Right Rotation): Це обертання виконується, коли незбалансованість виникає через зростання висоти правого піддерева правого нащадка кореня. Ось як це працює:

- Корінь правого піддерева стає новим коренем.
- Піддерево лівого сина кореня стає правим піддеревом попереднього кореня.
- Корінь дерева стає лівим сином нового кореня



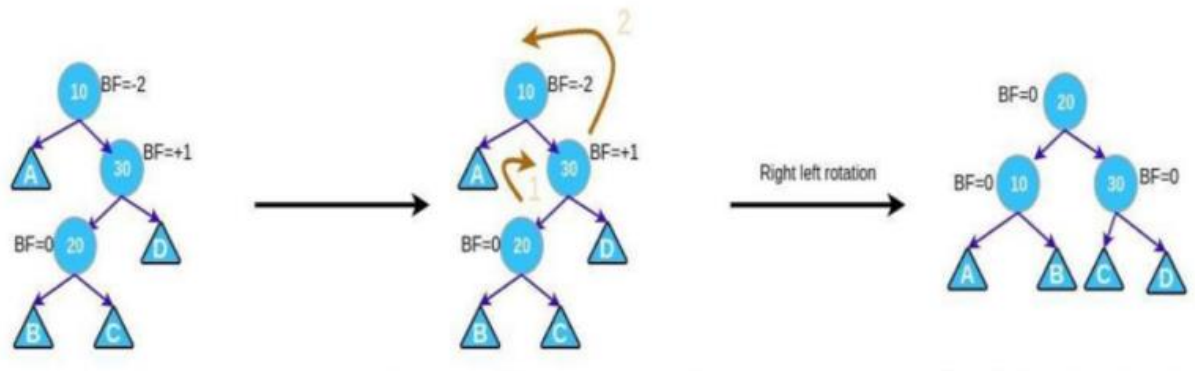
LR - Обертання вправо-вліво (Right-Left Rotation): Це обертання виконується, коли незбалансованість виникає через зростання висоти правого піддерева лівого нащадка кореня. Ось як це працює:

- Виконується праве обертання для правого піддерева.
- Потім виконується ліве обертання для дерева в цілому.



RL - Обертання вліво-вправо (Left-Right Rotation): Це обертання виконується, коли незбалансованість виникає через зростання висоти лівого піддерева правого нащадка кореня. Ось як це працює:

- Виконується ліве обертання для лівого піддерева.
- Потім виконується праве обертання для дерева в цілому



Виконання лабораторної роботи:

1. Розробка структури AVL-дерева:

У першому етапі була створена структура даних AVL-дерева. Ця структура включала наступні поля: ключ вузла, вказівники на лівого та правого нащадків, висота вузла.

2. Реалізація основних операцій:

Далі були реалізовані основні операції з AVL-деревом:

- newNode: Створення нового вузла з заданим ключем.
- rightRotate, leftRotate: Операції правого та лівого обертання.
- insert: Вставка нового вузла в AVL-дерево зі збереженням балансу.
- deleteNode: Видалення вузла з AVL-дерева зі збереженням балансу.
- search: Пошук вузла з заданим ключем в AVL-дереві.

3. Виведення AVL-дерева:

Також було реалізовано функцію printTree, яка виводила AVL-дерево у вигляді дерева з використанням символів "/", "" та "-" для відображення структури дерева.

4. Тестування правильності роботи:

Для перевірки правильності роботи функцій були написані тести з використанням Google Test Framework. Тести перевіряли різні аспекти роботи функцій, включаючи вставку, видалення та пошук вузлів, обчислення висоти та балансу дерева, праве та ліве обертання дерева.

Результати роботи програмної реалізації структури AVL-дерева:

Приклад №1:

```
AVL tree is:
|       /-- 50
|   /-- 40
\-- 30
    |   /-- 25
    \-- 20
        \-- 10
```

```
Preorder traversal of the constructed AVL tree is
30 20 10 25 40 50

Deleting  20
AVL tree after deletion of 20 is
|       /-- 50
|   /-- 40
\-- 30
    \-- 25
        \-- 10
Preorder traversal of the AVL tree after deletion of 20
30 25 10 40 50

Search 30
Key 30 exists in AVL tree
Search 3100
Key 3100 does not exist in AVL tree
```

Приклад 2:

```
 AVL tree is:
|               /-- 50
|           /-- 40
|       /-- 30
|       |   \-- 25
|   /-- 20
|   |   |           /-- 17
|   |   |       /-- 16
|   |   |   /-- 15
|   |   |   |   \-- 14
|   |   \-- 13
|   |       \-- 12
|   |           \-- 11
\-- 10
    |           /-- 9
    |       /-- 8
    |       |   \-- 7
    |   /-- 6
    |   |   \-- 5
    \-- 4
        |   /-- 3
        \-- 2
            \-- 1
Preorder traversal of the constructed AVL tree is
10 4 2 1 3 6 5 8 7 9 20 13 12 11 15 14 16 17 30 25 40 50
```

Тестування

1. Тест висоти вузла (height)

```
TEST(AVLTreeTest, HeightTest) {
    Node* root = nullptr;
    EXPECT_EQ(height(root), 0);

    root = newNode(10);
    EXPECT_EQ(height(root), 1);
}
```

2. Тест максимального значення (max)

```
TEST(AVLTreeTest, MaxTest) {
    EXPECT_EQ(max(5, 10), 10);
    EXPECT_EQ(max(15, 3), 15);
    EXPECT_EQ(max(7, 7), 7);
}
```

3. Тест створення нового вузла (newNode)

```
TEST(AVLTreeTest, NewNodeTest) {
    Node* node = newNode(10);
    EXPECT_EQ(node->key, 10);
    EXPECT_EQ(node->left, nullptr);
    EXPECT_EQ(node->right, nullptr);
    EXPECT_EQ(node->height, 1);
}
```

4. Тест правого обертання (rightRotate)

```
TEST(AVLTreeTest, RightRotateTest) {
    Node* y = newNode(30);
    y->left = newNode(20);
    y->left->left = newNode(10);

    y = rightRotate(y);

    ASSERT_EQ(y->key, 20);
    ASSERT_EQ(y->right->key, 30);
    ASSERT_EQ(y->left->key, 10);
}
```

5. Тест лівого обертання (leftRotate)

```
TEST(AVLTreeTest, LeftRotateTest) {
    Node* x = newNode(10);
    x->right = newNode(20);
    x->right->right = newNode(30);

    x = leftRotate(x);

    ASSERT_EQ(x->key, 20);
    ASSERT_EQ(x->left->key, 10);
    ASSERT_EQ(x->right->key, 30);
}
```

6. Тест балансу вузла (balance)

```
TEST(AVLTreeTest, BalanceTest) {
    Node* root = nullptr;
    EXPECT_EQ(getBalance(root), 0);

    root = newNode(10);
    EXPECT_EQ(getBalance(root), 0);

    root->left = newNode(5);
    EXPECT_EQ(getBalance(root), 1);

    root->right = newNode(15);
    EXPECT_EQ(getBalance(root), 0);

    root->right->right = newNode(20);
    EXPECT_EQ(getBalance(root), 0);
}
```

7. Тест вставки та видалення вузла (insertionAndDeletion)

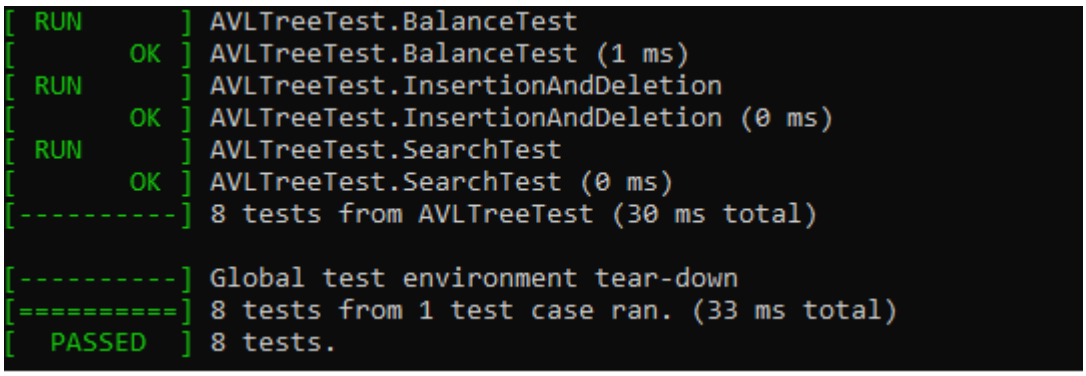
```
TEST(AVLTreeTest, InsertionAndDeletion) {
    Node* root = nullptr;

    root = insert(root, 10);
    root = insert(root, 20);
    root = insert(root, 30);
    root = insert(root, 40);
    root = insert(root, 50);
    root = insert(root, 25);

    root = deleteNode(root, 20);
    EXPECT_TRUE(search(root, 10));
    EXPECT_TRUE(search(root, 25));
    EXPECT_TRUE(search(root, 30));
    EXPECT_TRUE(search(root, 40));
    EXPECT_TRUE(search(root, 50));
    EXPECT_FALSE(search(root, 20));
    EXPECT_FALSE(search(root, 35));
}
```

9. Тест пошуку вузла (search)

```
TEST(AVLTreeTest, SearchTest) {
    Node* root = nullptr;
    EXPECT_FALSE(search(root, 10));

    root = newNode(10);
    root->left = newNode(5);
    root->right = newNode(15);
    root->left->left = newNode(3);
    root->left->right = newNode(7);

    EXPECT_TRUE(search(root, 10));
    EXPECT_TRUE(search(root, 5));
    EXPECT_TRUE(search(root, 15));
    EXPECT_TRUE(search(root, 3));
    EXPECT_TRUE(search(root, 7));

    EXPECT_FALSE(search(root, 1));
    EXPECT_FALSE(search(root, 8));
}
```

Результати тестів:

```
[==========] Running 8 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 8 tests from AVLTreeTest
[ RUN      ] AVLTreeTest.HeightTest
[       OK ] AVLTreeTest.HeightTest (0 ms)
[ RUN      ] AVLTreeTest.MaxTest
[       OK ] AVLTreeTest.MaxTest (0 ms)
[ RUN      ] AVLTreeTest.NewNodeTest
[       OK ] AVLTreeTest.NewNodeTest (0 ms)
[ RUN      ] AVLTreeTest.RightRotateTest
[       OK ] AVLTreeTest.RightRotateTest (0 ms)
[ RUN      ] AVLTreeTest.LeftRotateTest
[       OK ] AVLTreeTest.LeftRotateTest (0 ms)
```



Висновок: у результаті виконання лабораторної роботи я ознайомилася з AVL-деревами та реалізувала основні операції з ними. Також написала тести для перевірки правильності роботи функцій. AVL-дерева є ефективною структурою даних для зберігання та швидкого пошуку впорядкованих даних, а їх балансування дозволяє підтримувати оптимальну швидкодію операцій вставки, видалення та пошуку.