Київський національний університет імені Тараса Шевченка
Факультет комп`ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Алгоритми та складність

Варіант №9

“АА-дерево ”

Виконав студент 2-го курсу
Групи ІПС-21
Тесленко Назар Олександрович

Київ – 2025

Завдання:

Реалізувати AA-дерево (дійсні числа)

Теорія:

AA-дерево — це збалансована деревовидна структура даних, винайдена Арне Андерсоном. Це спрощена варіація червоно-чорного дерева, що зберігає баланс за допомогою рівнів (levels) та спеціальних операцій балансування.

Структура Node для AA-дерева:

- val — значення, що зберігається у вузлі (тип double)
- level — рівень вузла, ключовий для балансування дерева
- right, left — вказівники на правого та лівого нащадків
- parent — вказівник на батьківський вузол

nullNode - спеціальний вузол-стоп (sentinel node), що використовується замість nullptr для спрощення крайових випадків. У реалізації nullNode має рівень 0 та посилається на самого себе.

Правила AA-дерева:

- Кожен нелистовий вузол має двох нащадків
- AA-дерево — це бінарне дерево пошуку, де для кожного вузла всі значення у лівому піддереві менші за значення вузла, а у правому — більші
- Рівень листового вузла завжди дорівнює 1
- Рівень лівого нащадка завжди менший за рівень батька
- Рівень правого нащадка або дорівнює рівню батька, або менший на 1
- Правий нащадок правого нащадка не може мати той самий рівень, що й його дід (правило послідовних горизонтальних зв'язків)

Операції балансування:

Skew (горизонтальне обертання вправо)

Операція skew коригує горизонтальні лівосторонні зв'язки. Виконується, коли лівий нащадок має той самий рівень, що й батько. По суті, це обертання вправо для виправлення порушення правила AA-дерева.

Split (вертикальне обертання вліво)

Операція split коригує послідовні горизонтальні правосторонні зв'язки. Виконується, коли правий нащадок правого нащадка має той самий рівень, що й батько. Це обертання вліво з підвищенням рівня нового кореня піддерева.

Операції з деревом:

Insert (вставка)

Процес додавання нового елемента до дерева. Спочатку новий вузол вставляється за правилами бінарного дерева пошуку, потім виконуються операції skew та split для збереження балансу дерева.

Remove (видалення)

Процес видалення елемента з дерева.

Включає різні випадки:

- Видалення листового вузла
- Видалення вузла з одним нащадком
- Видалення вузла з двома нащадками (з пошуком наступника)

Find (пошук)

Стандартний рекурсивний пошук по бінарному дереву, що повертає вузол з шуканим значенням або nullNode, якщо значення не знайдено.

Алгоритм:

Ініціалізація дерева

- Створюється спеціальний nullNode
- Вказівники left і right у nullNode посилаються на себе
- Корінь дерева ініціалізується як nullNode

Операція вставки (insert)

- Якщо дерево порожнє, створюється новий вузол з рівнем 1, який стає коренем
- В іншому випадку, рекурсивно знаходиться місце для вставки за правилами бінарного дерева пошуку:
    - Якщо значення менше за поточний вузол, рекурсивно переходимо до лівого піддерева
    - Якщо значення більше, рекурсивно переходимо до правого піддерева
    - Якщо значення дорівнює, це дублікат і вставка припиняється

- Після вставки виконуються операції балансування:
    - skew для виправлення горизонтальних лівих зв'язків
    - split для виправлення послідовних правих горизонтальних зв'язків

Операція skew (корекція лівих горизонтальних зв'язків)

- Якщо лівий нащадок має той самий рівень, що й поточний вузол, виконується ротація:
    - Лівий нащадок стає новим коренем піддерева
    - Правий нащадок лівого нащадка стає лівим нащадком старого кореня
    - Старий корінь стає правим нащадком нового кореня
    - Оновлюються вказівники на батьківські вузли
- Рівні вузлів залишаються незмінними

Операція split (корекція послідовних правих горизонтальних зв'язків)

- Якщо правий нащадок правого нащадка має той самий рівень, що й поточний вузол:
    - Правий нащадок стає новим коренем піддерева
    - Лівий нащадок правого нащадка стає правим нащадком старого кореня
    - Старий корінь стає лівим нащадком нового кореня
    - Оновлюються вказівники на батьківські вузли
    - Рівень нового кореня збільшується на 1

Операція видалення (remove)

- Рекурсивно шукається вузол для видалення за правилами бінарного дерева пошуку
- Коли вузол знайдено, обробляється один з варіантів:
    - **Випадок 1:** Якщо вузол є листком (не має нащадків) - просто видаляється
    - **Випадок 2.1:** Якщо вузол має лише правого нащадка - вузол замінюється правим нащадком
    - **Випадок 2.2:** Якщо вузол має лише лівого нащадка - вузол замінюється лівим нащадком
    - **Випадок 3:** Якщо вузол має обох нащадків:
        - Знаходиться наступник вузла (найменший вузол у правому піддереві) методом findMin
        - Значення вузла замінюється значенням наступника
        - Наступник видаляється з правого піддерева рекурсивним викликом remove

- Після видалення виконується перебалансування дерева методом rebalance

Операція перебалансування (rebalance)

- Визначається необхідний рівень вузла як мінімальний рівень його нащадків плюс 1
- Якщо поточний рівень вузла більший за необхідний:
    - Рівень вузла знижується до необхідного
    - Якщо рівень правого нащадка більший за необхідний, його рівень також знижується
- Виконуються операції skew для поточного вузла та його правого піддерева (включно з правим нащадком правого нащадка)
- Виконуються операції split для поточного вузла та його правого нащадка

Операція пошуку (find)

- Рекурсивно шукається вузол за правилами бінарного дерева пошуку:
    - Якщо поточний вузол - nullNode, значення не знайдено, повертається nullNode
    - Якщо значення менше за поточний вузол, рекурсивно пошук продовжується в лівому піддереві
    - Якщо значення більше за поточний вузол, рекурсивно пошук продовжується в правому піддереві
    - Якщо значення дорівнює значенню поточного вузла, повертається цей вузол
- Публічний метод find перетворює результат у булеве значення, порівнюючи повернений вузол з nullNode

Мова реалізації: C++

Модулі програми

- **class AATree**

Реалізує структуру даних AA-дерево, яке є збалансованою варіацією бінарного дерева пошуку із спрощеними правилами балансування порівняно з червоно-чорним деревом. Використовує концепцію рівнів для підтримки збалансованості.

Головні методи:

- **void insert(double val)** – додає новий елемент зі значенням val у дерево. Виконує балансування за допомогою операцій skew та split.
- **void remove(double val)** – видаляє елемент зі значенням val із дерева, коригує структуру дерева та виконує перебалансування.
- **bool find(double val)** – перевіряє наявність елемента зі значенням val у дереві, повертає true, якщо елемент знайдено.
- **void print()** – виводить дерево у вигляді вкладеної структури, відображаючи його поточний стан з рівнями вузлів.

Допоміжні приватні методи:

- *void skew(Node& node)* – виконує праве обертання для виправлення горизонтальних лівих зв'язків
- *void split(Node& node)* – виконує ліве обертання для виправлення послідовних правих горизонтальних зв'язків
- *void insert(double val, Node& node, Node* parent) – рекурсивна функція вставки з модифікацією дерева та балансуванням.
- *void remove(double val, Node& node)*– рекурсивне видалення елемента з дерева.
- *Node find(double val, Node* node) – рекурсивний пошук елемента в дереві, повертає вказівник на знайдений вузол або nullNode.
- *void rebalance(Node& node)* – перебалансовує дерево після видалення елемента, коригуючи рівні та структуру.
- *Node findMin(Node* node)– знаходить вузол з найменшим значенням у піддереві, використовується при видаленні.
- *void print(Node node, int space)* – рекурсивний вивід структури дерева з відступами.

Складність алгоритму AA-дерева та залежних методів

Головні методи:

Вставка елемента insert(val) – O(log n)

- Рекурсивний пошук місця вставки в дереві – O(log n)

- Створення нового вузла – O(1)
- Виконання операції skew – O(1)
- Виконання операції split – O(1)

Пошук елемента find(val) – O(log n)

- Рекурсивний пошук у бінарному дереві – O(log n)
- Перевірка на наявність елемента – O(1)

Видалення елемента remove(val) – O(log n)

- Рекурсивний пошук вузла для видалення – O(log n)
- Обробка випадків видалення:
    - Видалення листка – O(1)
    - Видалення вузла з одним нащадком – O(1)
    - Видалення вузла з двома нащадками:
        - Знаходження наступника (findMin) – O(log n)
        - Заміна вузла наступником – O(1)
        - Видалення наступника – O(log n)
- Перебалансування дерева (rebalance) – O(log n)

Виведення дерева print() – O(n)

- Рекурсивний обхід всіх вузлів дерева – O(n)
- Виведення значення і рівня кожного вузла – O(1)

Допоміжні методи:

Операція skew (лівий поворот) – O(1)

- Перевірка умови для виконання skew – O(1)
- Зміна батьківських посилань та оновлення зв'язків – O(1)
- Оновлення посилань на батьків – O(1)

Операція split (правий поворот) – O(1)

- Перевірка умови для виконання split – O(1)

- Зміна батьківських посилань та оновлення зв'язків – O(1)
- Оновлення посилань на батьків – O(1)
- Збільшення рівня нового кореня піддерева – O(1)

Перебалансування дерева rebalance(node) – O(1)

- Визначення необхідного рівня вузла – O(1)
- Коригування рівнів вузлів – O(1)
- Виконання операцій skew (для поточного вузла і нащадків) – O(1)
- Виконання операцій split (для поточного вузла і нащадків) – O(1)

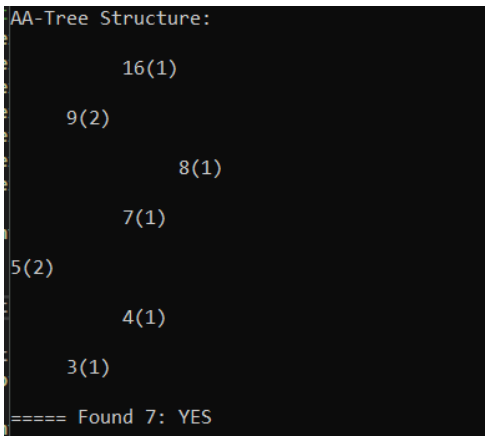
Знаходження мінімального вузла findMin(node) – O(log n)

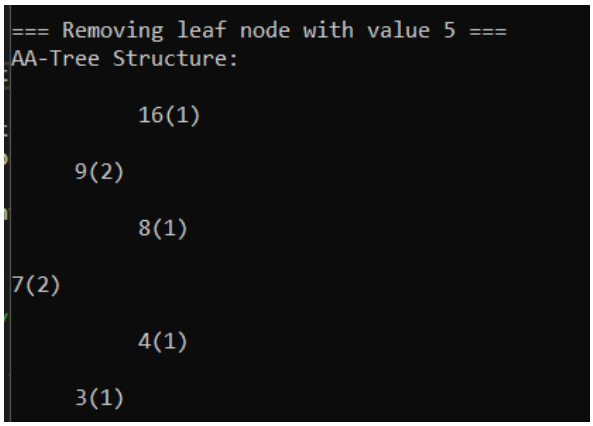
- Ітерація вліво до крайнього лівого вузла – O(log n)
- Повернення знайденого вузла – O(1)

Тестові приклади:

Тест 1 (натуральні числа)

```
////////////////TEST 1/////////////////
AATree tree;

//Elements insertion
tree.insert(5);
tree.insert(4);
tree.insert(9);
tree.insert(16);
tree.insert(3);
tree.insert(8);
tree.insert(7);

tree.print();


std::cout << "Found 7: " << (tree.find(7) ? "YES" : "NO(((") << std::endl;

std::cout << "=== Removing leaf node with value 5 ===\n";
tree.remove(5);

tree.print();

return 0;
```



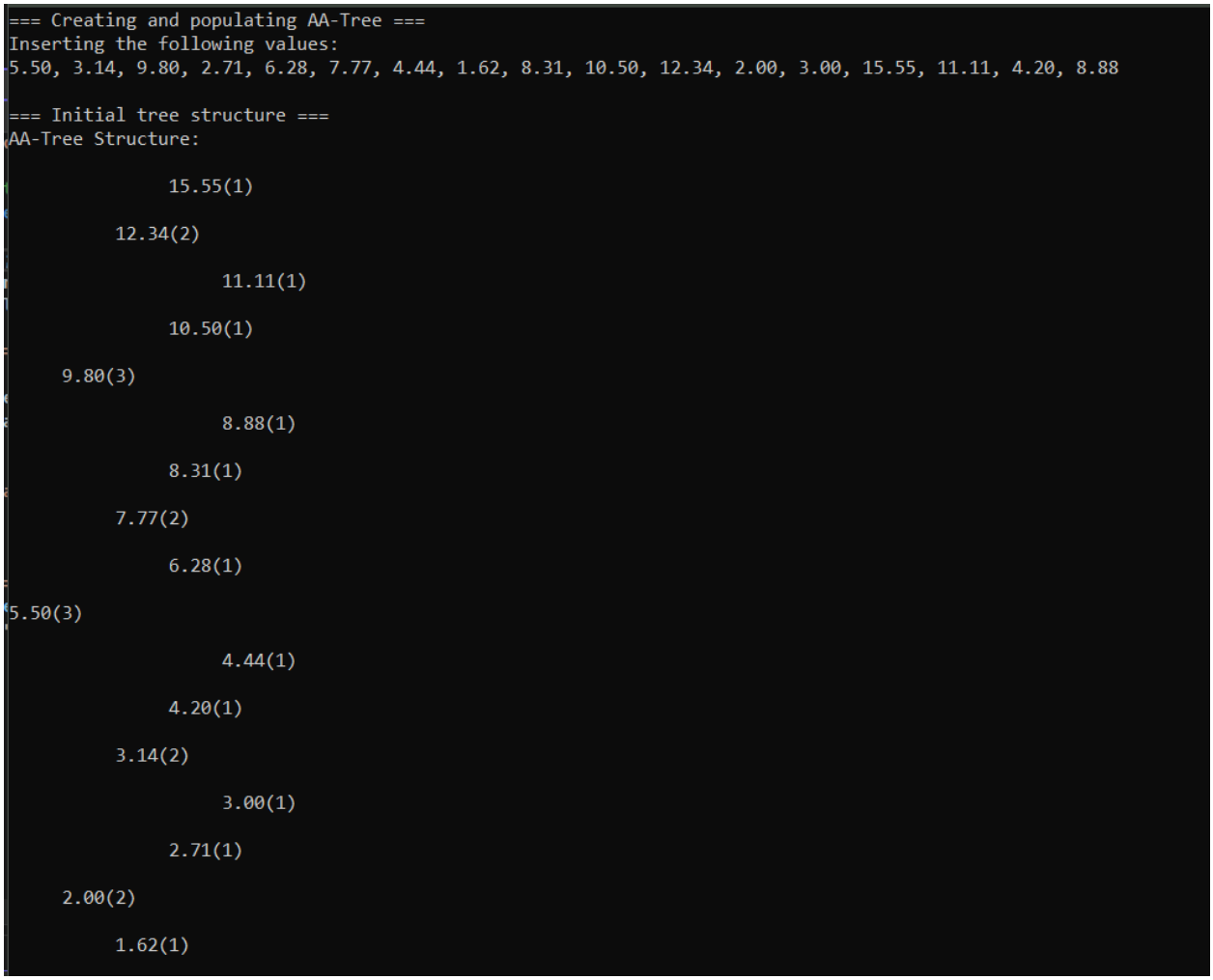


Отже, як можемо бачити головний функціонал АА-дерева працює, а саме: коректна вставка вузлів, ребалансування, пошук взулів з певним значенням та видалення. Отже для натуральних числел алгоритм відпрацював вірно.

Тест 2 (Раціональні числа)

Створюємо дерево з такого вектору вузлів:

{ 5.5, 3.14, 9.8, 2.71, 6.28, 7.77, 4.44, 1.618, 8.31, 10.5, 12.34, 2.0, 3.0, 15.55, 11.11, 4.2, 8.88 }



```
=== Searching for element 7.77 ===
Result: Found

=== Searching for element 13.13 ===
Result: Not found

=== Searching for edge element 15.55 ===
Result: Found
```

Влаштовуємо пошук елементів за трьома випадками:

- Вершина існує
- Вершина не існує
- Вершина – лситок

Як можемо бачити алгоритм пошуку спрацював праивльно

Видалення елементів:

- **Випадок видалення листкового елементу:**

```
=== Removing leaf node 1.62 ===
Tree structure after removal:
AA-Tree Structure:

              15.55(1)

         12.34(2)

                   11.11(1)

              10.50(1)

    9.80(3)

                   8.88(1)

              8.31(1)

         7.77(2)

              6.28(1)

5.50(3)

                   4.44(1)

              4.20(1)

         3.14(2)

              3.00(1)

    2.71(2)

         2.00(1)
Removal check: Element successfully removed
```

- **Випадок видалення батька з одним нащадком**

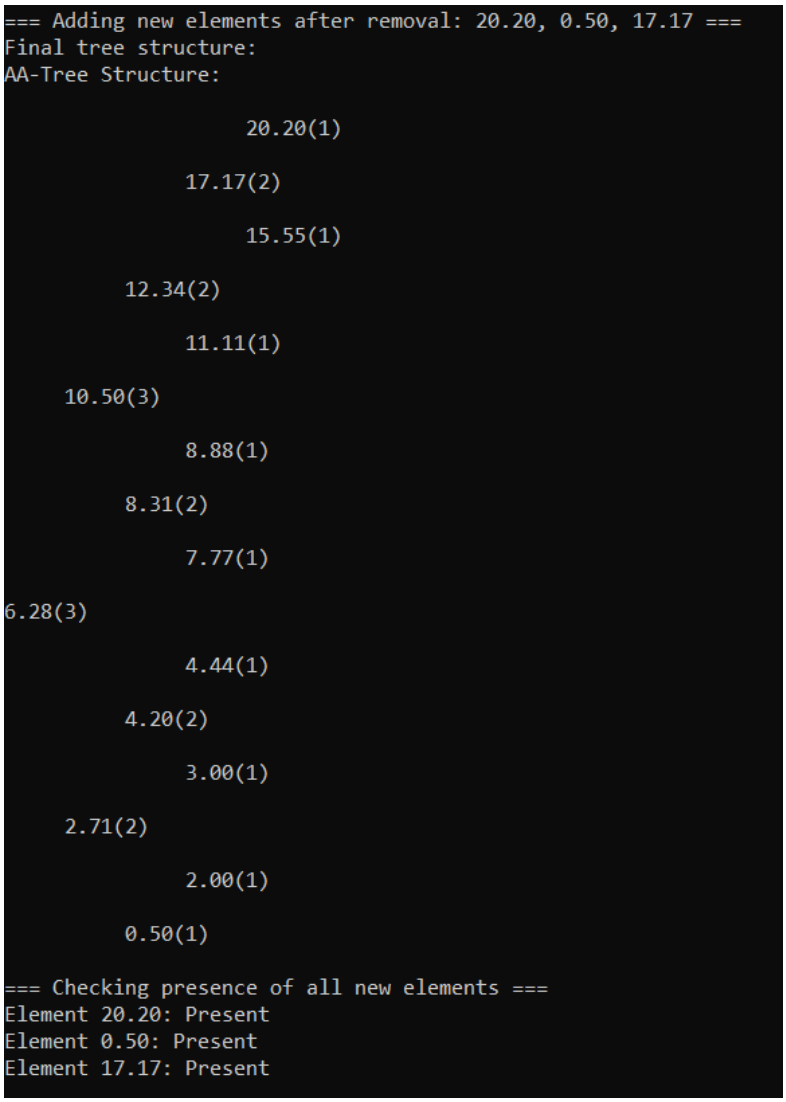
```
=== Removing node with one child 3.14 ===
Tree structure after removal:
AA-Tree Structure:

              15.55(1)

         12.34(2)

                   11.11(1)

              10.50(1)

     9.80(3)

                   8.88(1)

              8.31(1)

         7.77(2)

              6.28(1)

5.50(3)

              4.44(1)

         4.20(2)

              3.00(1)

     2.71(2)

         2.00(1)
Removal check: Element successfully removed
```

- **Випадок видалення батька з обома нащадками**

```
=== Removing node with two children 5.50 ===
Tree structure after removal:
AA-Tree Structure:

              15.55(1)

         12.34(2)

                   11.11(1)

              10.50(1)

     9.80(3)

              8.88(1)

         8.31(2)

              7.77(1)

6.28(3)

              4.44(1)

         4.20(2)

              3.00(1)

     2.71(2)

         2.00(1)
Removal check: Element successfully removed
```

Для всіх трьох випадків елемнти були видалені алгоритмічно і візуально коректно

Вставка елемнтів після видалення:



Елементи були вірно додані до дерева а успішно знайдені.

Висновок:

У даній роботі було реалізовано АА-дерево, яке є збалансованою структурою даних, що поєднує простоту реалізації зі ефективністю операцій бінарного дерева пошуку. Запропонована реалізація забезпечує логарифмічну складність для основних операцій вставки, пошуку та видалення елементів, що є оптимальним для роботи з наборами даних різного об'єму.

Особливістю АА-дерева є використання концепції рівнів вузлів замість кольорів, як у червоно-чорному дереві, та лише двох операцій балансування: skew і split. Це спрощує підтримку збалансованості структури даних при модифікаціях та гарантує ефективний доступ до елементів навіть при активних змінах вмісту дерева.

Використані джерела:

- Лекція 3 з АіС
- https://en.wikipedia.org/wiki/AA_tree
- https://www.geeksforgeeks.org/aa-trees-set-1-introduction/
- https://cs.valdosta.edu/~dgibson/courses/cs3410/notes/ch19_6.pdf