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

Лабораторна робота №2_1
“Реалізація АА дерева”
Виконав студент 2-го курсу
Групи ІПС-22
Левицький Іван Олегович

2025

Завдання: реалізувати побудову АА дерева, додавання і видалення елемента в це дерево, щоб воно зберігало свої властивості, і виведення цього дерева для типу даних**: Раціональні числа**

Предметна область: Структура зберігання даних і їх оптимальний пошук

Теорія:

АА-дерево (AA-tree) — це варіант **самобалансного бінарного дерева пошуку (BST)**, який забезпечує ефективні операції:

- пошук
- вставку
- видалення

всі за **O(log n)** у найгіршому випадку.

Це спрощена версія **RB-дерева**, де баланс підтримується завдяки спеціальній структурі рівнів та обмеженим правилам.

Основна ідея

- Кожна вершина має "рівень" (аналог кольору в червоно-чорних деревах).
- Рівень листа = 1, порожні вузли вважаються рівня 0.
- Порушення балансу виправляються за допомогою лише двох операцій:
  - skew (вирівнювання)
  - split (розщеплення)

Основні правила (інваріанти)

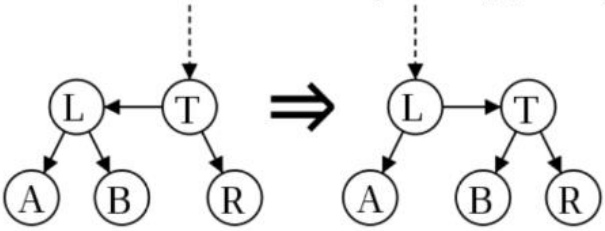
Щоб дерево було коректним, мають виконуватись такі умови:

- Лівий син завжди має рівень < рівня батька
  (лівий "червоний" син — заборонено!)
- Правий син може мати той самий рівень, але не більший
  (тобто може бути "червоним")
- Не можна мати двох правих синів поспіль на одному рівні
  (заборонені подвійні праві "червоні" ребра)
- Усі листові вузли мають рівень 1, а порожні (NULL) — рівень 0
- Рівень вузла не може бути меншим за обидва рівні його дітей (інакше skew)

skew(x)

*Якщо у вузла x є **лівий син того ж рівня**, ми робимо **праве обертання.*** Це виправляє порушення правила, де лівий син має такий самий рівень — а це заборонено.

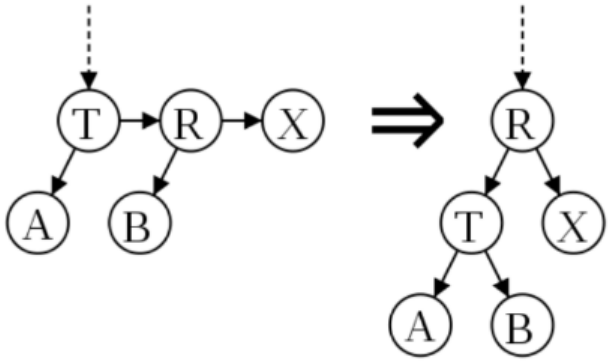
SKEW (усунення лівого зв’язку на одному рівні)



split(x)

Якщо у вузла x є **правий син**, і той має ще одного **правого сина** на тому ж рівні — робимо **ліве обертання**, і **збільшуємо рівень** нового батька на 1. Це усуває подвійне праве ребро (дві червоні стрілки поспіль).

SPLIT (усунення двох правих зв’язків на одному рівні).



Переваги:

- Код вставки / видалення простіший, ніж у червоно-чорному дереві

- Тільки два випадки, які потрібно перевірити (skew, split)

- Висота дерева гарантується як $\mathbf{\log_2(n)}$

Алгоритми:

Алгоритм вставки:

1. **Як в звичайному BST** — рекурсивно шукаємо, куди вставити.
2. **Вставляємо вузол з рівнем 1**.
3. **Балансуємо**:
    - Спочатку викликаємо skew (усуває лівого червоного сина)
    - Потім split (усуває два правих червоних сина)

Алгоритм видалення:

1. **Як у BST** — шукаємо вузол для видалення.
2. Якщо знайдений:
    - Якщо **вузол має двох дітей** — замінюємо значення на мінімальне з правого піддерева (in-order successor), і далі видаляємо дубліката.
    - Якщо **один або жодного** — видаляємо просто.

3. **Балансування**:
    - **Зменшуємо рівень** вузла, якщо потрібно (правило підтримки рівнів)
    - Потім викликаємо:
        - skew для node, node.right, node.right.right
        - split для node, node.right

Складність:

Як і було зазначено вище середня і найгірша складність для алгоритму вставки, видалення і пошуку елемента в AA дереві складає **O(log n).**

Мова програмування:
C++.

Модулі програми:

class Rational – клас, який виконує функціонал раціонального числа:

int numerator; - чисельник
int denominator; - знаменник

і містить операції: +, -, *, /, ==, !=, <, >, <=, >=, і перетворення в рядок(toString()).

struct Node {
```
    Rational value;
    Node* left;
    Node* right;
    int level;

    Node(const Rational& val)
        : value(val), left(nullptr), right(nullptr), level(1) {
    }
```
}; - клас, який реалізовує вузол нашого AA дерева

class AATree – реалізація нашого AA дерева

Node* root = nullptr; з даних містить тільки корінь.

Node* skew(Node* node) – метод, який реалізовує 1 із 2 базових методів AA дерева, тобто якщо з'явився лівий червоний син (тобто з рівнем, як у батька) → робимо праве обертання

1)**Перевірка вхідних умов:**

- Спочатку метод перевіряє, чи існує поточний вузол (node) або його лівий нащадок (node->left). Якщо хоча б один з них є null, метод повертає вхідний вузол без змін. Це запобігає спробі обробки вузлів, яких не існує.

2)**Перевірка рівнів:**

- Якщо рівень вузла node збігається з рівнем його лівого нащадка (node->left->level), це означає, що є порушення правил AA-дерева, і потрібне балансування. Зокрема, це може статися через те, що лівий вузол знаходиться занадто високо, що створює "перекіс" (skew).

3)**Ротація вправо (правий поворот):**

- Лівий нащадок вузла node (назвемо його L) стає новим коренем піддерева.
- Праве піддерево вузла L (L->right) стає новим лівим піддеревом для вузла node.
- Вузол node стає правим піддеревом для вузла L.

4)**Повернення нового кореня:**

- Новий корінь (L) після правого повороту повертається як результат роботи функції. Це означає, що піддерево тепер збалансоване.

5)**Завершення:**

- Якщо перевірка рівнів не виявила порушення, метод просто повертає вузол node

Node* split(Node* node) -

1)**Перевірка умов коректності:**

Спочатку перевіряється, чи є вхідний вузол node або його праве піддерево (або його праве піддерево на рівень нижче) null. Якщо будь-яка з цих умов виконується, метод повертає node, оскільки операція балансування не потрібна.

2)**Перевірка умови рівнів:**

Метод перевіряє, чи рівень вузла node співпадає з рівнем його правого-внука (node->right->right->level). Якщо це так, це означає, що баланс порушений, і виконується операція "повороту" (rotation).

3)**Поворот ліворуч:**

- Вузол R (правий нащадок node) стає новим коренем піддерева.
- Праве піддерево вузла node (тобто R->left) стає новим правим піддеревом для node.
- Вузол node стає лівим піддеревом для R.

4)**Збільшення рівня вузла:**

Рівень нового кореня R збільшується на 1, щоб відобразити зміну структури дерева.

5)**Повернення нового кореня:**

Після проведення повороту функція повертає новий корінь піддерева R, який тепер є збалансованим.

6)**Завершення в інших випадках:**

Якщо умова рівнів не виконується, повертається оригінальний вузол node без змін.

insert(Node* node, const Rational& value):

Метод для вставки нового значення в AA-дерево:

- Якщо вузол node порожній, створюється новий вузол з заданим значенням.
- Рекурсивно перевіряється, чи потрібно вставити значення в ліве або праве піддерево.

- Якщо значення вже існує у дереві, повертається поточний вузол (дублікати не додаються).
- Виконується балансування за допомогою методів skew і split.

inOrder(Node* node) const:

Метод для обходу дерева в порядку інфіксного обходу:

- Рекурсивно виконується обхід лівого піддерева, поточного вузла та правого піддерева.
- Друкує значення вузлів в порядку зростання.

clear(Node* node):

Метод для очищення дерева:

- Рекурсивно видаляє всі вузли дерева, проходячи ліве і праве піддерево.
- Виконує операцію delete для кожного вузла.

printTree(Node* node, int indent = 0) const:

Метод для друку дерева у вигляді текстового представлення:

- Рекурсивно друкує вузли дерева з урахуванням рівня вкладеності (відступів).
- Відображає значення вузла та його рівень у дереві.

nodeID(Node* node) const:

Метод для генерації унікального ідентифікатора вузла:

- Використовує адресу вузла як його унікальний ідентифікатор.
- Повертає рядок для використання в інших методах.

exportDOT(Node* node, std::ofstream& out) const:

Метод для експорту дерева у формат DOT:

- Рекурсивно додає вузли та зв'язки дерева до файлу у форматі DOT.
- Маркує червоні ребра (відображає особливі властивості AA-дерева).
- Мета: Експорт дерева для його візуалізації за допомогою програм типу Graphviz.

decreaseLevel(Node* node):

Метод для зменшення рівня вузла:

- Обчислює очікуваний рівень вузла на основі рівнів його дітей.
- Якщо поточний рівень більший за очікуваний, встановлює новий рівень.
- Мета: Підтримка правил рівнів AA-дерева

remove(Node* node, const Rational& val):

Метод для видалення вузла з дерева:

- Рекурсивно шукає вузол з заданим значенням.
- Якщо вузол знайдений, видаляє його та виконує операції decreaseLevel, skew і split для балансування дерева.

validate(Node* node, int& height, std::string& error, const Rational* min = nullptr, const Rational* max = nullptr) const:

Метод для перевірки коректності дерева:

- Перевіряє, чи всі правила бінарного дерева пошуку та AA-дерева виконуються.
- Рекурсивно перевіряє рівні вузлів та послідовність значень.
- Мета: Забезпечити коректність структури дерева.

contains(Node* node, const Rational& value) const:

Метод для перевірки наявності значення у дереві:

- Рекурсивно шукає вузол з заданим значенням.
- Повертає true, якщо значення знайдено, або false, якщо його немає.
- Мета: З'ясувати, чи існує певне значення у дереві.

void generateTreeImage(std::string dotFilename):

функція, яка перетворює .dot файл в .png файл, який користувач може подивитись.

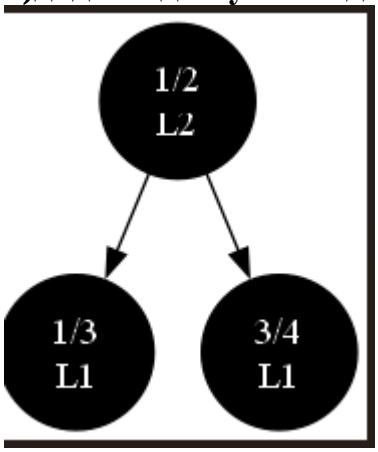
void openFile(const std::string& filename):

Функція, яка відкриває .png файл через код.

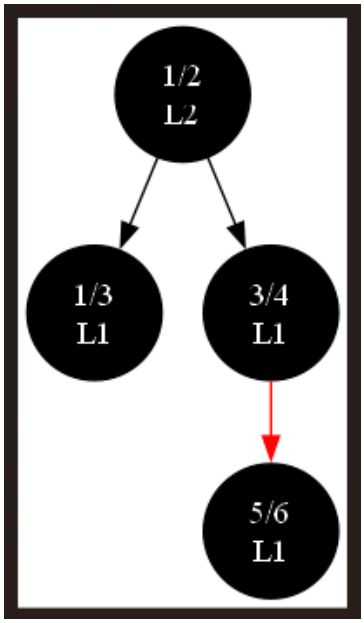
Інтерфейс користувача:

Користувач може додавати, видаляти і шукати певні елементи через команди в програмі, тобто **Консольний інтерфейс** , а також генерувати .png зображення, які будуть зображати AA дерево, яке користувач має на певному кроці, ці .png файли зберігаються на компютері в папці де зберігається і сама програма.

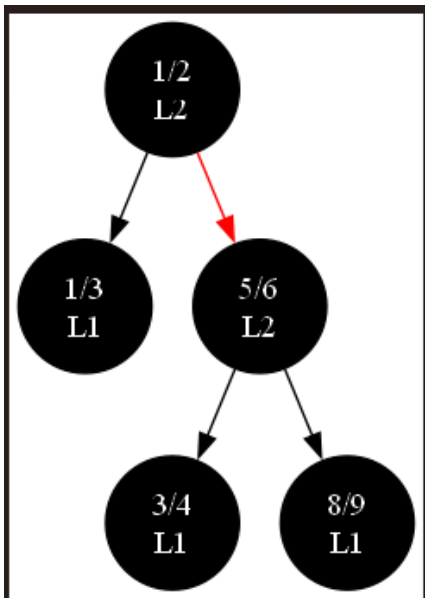
Тестові приклади:
1: - Приклад для пояснення
1)додаєм до пустого дерева 3 елементи ½, 1/3, 3/4



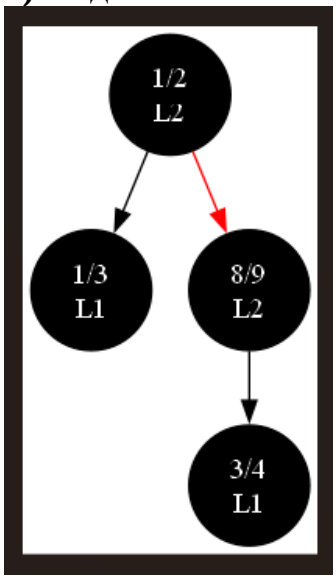
2) додаємо ще два елементи 5/6



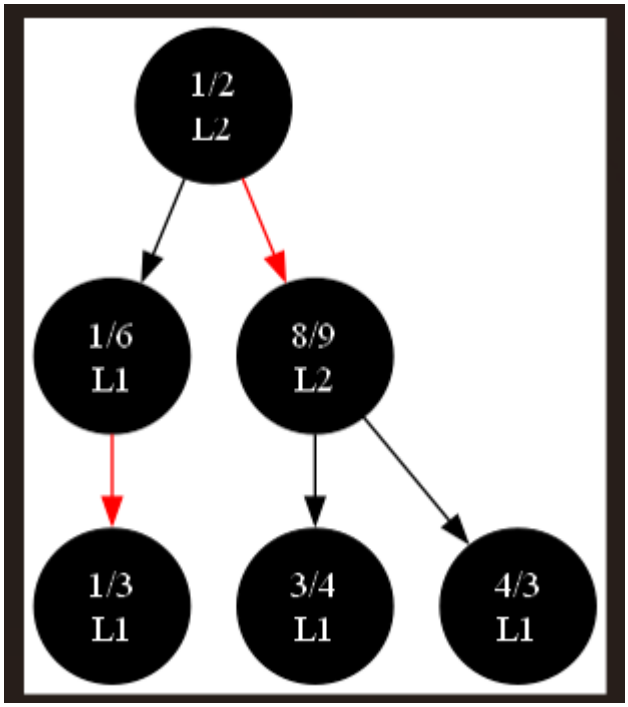
І 8/9



3) Видаляємо елемент 5/6



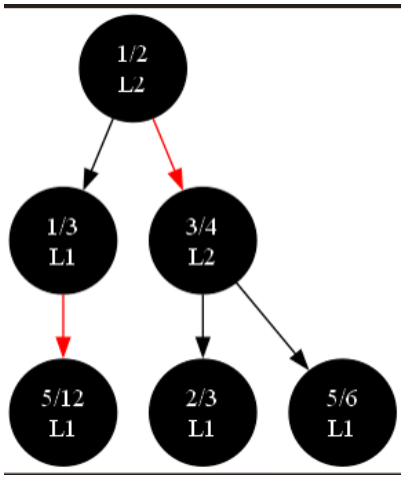
4) додаємо ще 2 елемента: 1/6 і 12/9



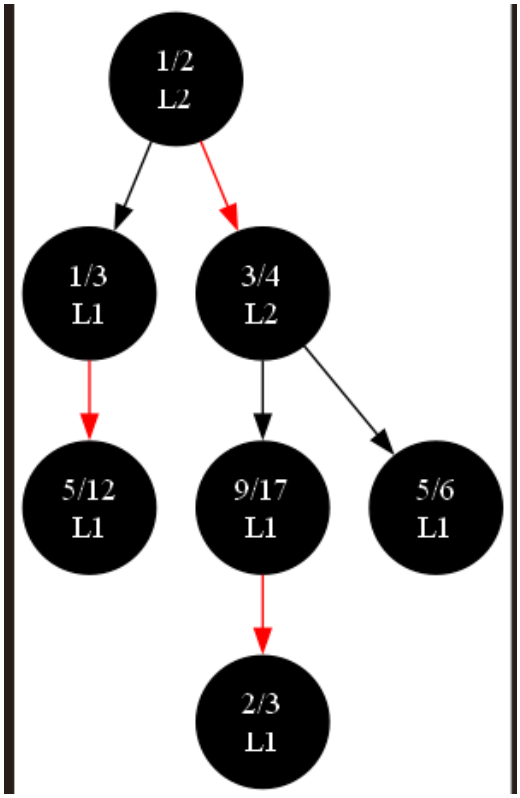
Другий приклад

1)додаємо певні значення:

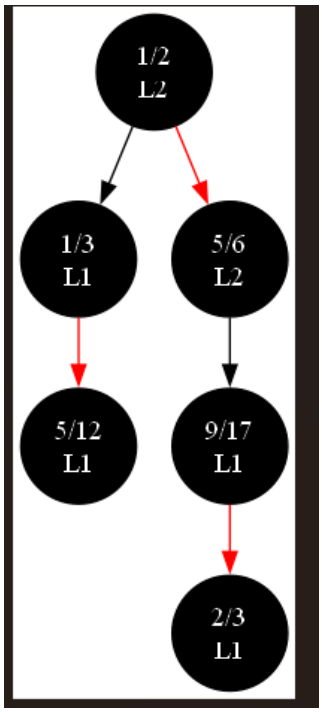
```
AATree tree;
tree.insert(val: Rational(num: 1, denom: 2));
tree.insert(val: Rational(num: 3, denom: 4));
tree.insert(val: Rational(num: 1, denom: 3));
tree.insert(val: Rational(num: 5, denom: 6));
tree.insert(val: Rational(num: 2, denom: 3));
tree.insert(val: Rational(num: 5, denom: 12));
```



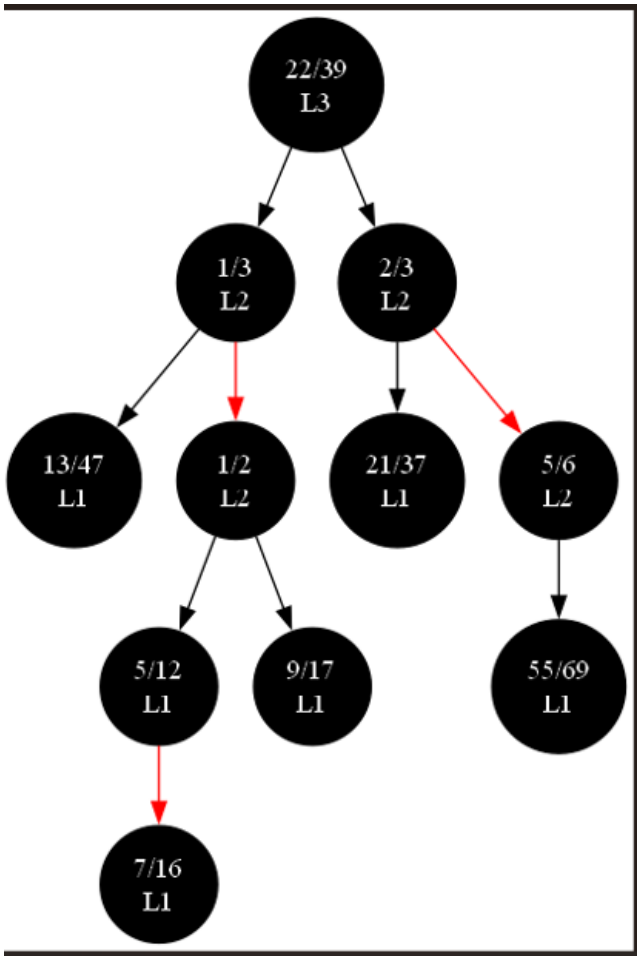
2) `tree.insert(Rational(18, 34));` - додаємо вузл 18/34 а тобто - 9/17



3) `tree.remove(Rational(3, 4));- Видаляємо вузл ¾`



4)
```
tree.insert(Rational(22, 39));
tree.insert(Rational(13, 47));
tree.insert(Rational(14, 32));
tree.insert(Rational(55, 69));
tree.insert(Rational(21, 37)); - додаємо ще 5 вузлів
```



Висновки: в даній лабораторній роботі було реалізовано АА дерево, а тобто вставка елемента в нього, видалення елемента з дерева, пошук елемнта в дереві, а також виведення АА дерева. Було побачено основний алгоритм роботи АА дерева, його переваги і недоліки.

Література:

1) https://en.wikipedia.org/wiki/AA_tree

2) https://web.eecs.umich.edu/~sugih/courses/eecs281/f11/lectures/12-AAtrees+Treaps.pdf

3) https://youtu.be/cRdon2QWgL4?si=rg_2oP7-oPg1oNMQ

4) Лекція 4 з предмету Алгоритми і складність