Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Лабораторна робота №2-2

“Реалізувати дерево відрізків (реалізація на основі червоно-чорного дерева).”

Виконала студентка 2-го курсу

Групи ІПС-22

Челушкіна Валерія Олександрівна

Київ – 2025

**Завдання**

Реалізація дерева відрізків (на основі червоно-чорного дерева) для раціональних чисел.

**Теорія.**

Дерево відрізків — це структура даних дерева, яка використовується для зберігання інформації про інтервали у вигляді дерева. Це дозволяє ефективно відповідати на запити діапазону над масивом і все ще є достатньо гнучким, щоб дозволити швидку зміну масиву. Дерево відрізків має потужні можливості та підтримує такі операції, як знаходження суми інтервалу, знаходження максимуму інтервалу, оновлення інтервалу та оновлення окремої точки.

У класичному варіанті дерево відрізків будується у вигляді бінарного дерева, де кожен вузол відповідає певному діапазону індексів масиву. Листові вузли зберігають значення елементів, а внутрішні — певну інформацію для певного діапазону.

Червоно-чорне дерево — різновид самозбалансованого бінарного дерева пошуку, вершини якого мають додаткові властивості, зокрема «колір» (червоний або чорний). Ці біти кольору використовуються для забезпечення того, щоб дерево залишалося приблизно збалансованим при виконанні операцій вставки та видалення.

Основні властивості червоно-чорного дерева:

* кожна вершина або червона, або чорна
* корінь дерева — чорний
* кожний лист (NIL) — чорний
* якщо вершина червона, обидві її дочірні вершини чорні (інакше, батько червоної вершини — чорний)
* усі прості шляхи від будь-якої вершини до листів мають однакову кількість чорних вершин

**Алгоритм.**

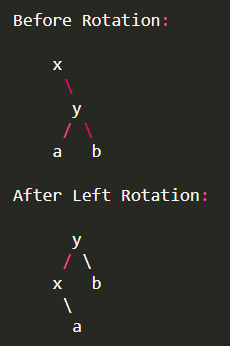
Розглянемо основні операції, потрібні для створення червоно-чорного дерева.

* Повороти

Повороти є основними операціями для підтримки збалансованої структури червоно-чорного дерева. Вони допомагають зберегти властивості дерева, гарантуючи, що найдовший шлях від кореня до будь-якого листа не більше ніж вдвічі перевищує довжину найкоротшого шляху. Обертання буває двох типів: ліворуч і праворуч.

* *Лівий поворот*

Лівий поворот у вузлі 𝑥 переміщує 𝑥 вниз ліворуч, а його правий дочірній елемент 𝑦 вгору, щоб зайняти місце 𝑥.

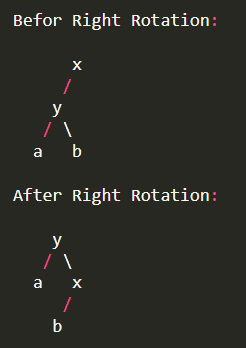


Кроки:

1. Встановіть *y* як правого дочірнього елемента *x*.
2. Перемістіть ліве піддерево *y* у праве піддерево *x*.
3. Оновіть батьківський елемент *x* і *y*.
4. Оновіть батьківський елемент *x*, щоб він вказував на *y* замість *x*.
5. Встановити лівий дочірній елемент *y* на *x*.
6. Оновіть батьківський елемент *x* на *y*.

* *Правий поворот*

Правий поворот у вузлі 𝑥 переміщує 𝑥 вниз праворуч, а його лівий дочірній елемент 𝑦 вгору, щоб зайняти місце 𝑥.



Кроки:

1. Встановіть *y* як лівого дочірнього елемента *x*.
2. Перемістіть праве піддерево *y* до лівого піддерева *x*.
3. Оновіть батьківський елемент *x* і *y*.
4. Оновіть батьківський елемент *x*, щоб він вказував на *y* замість *x*.
5. Установіть праву дочірню частину *y* на *x*.
6. Оновіть батьківський елемент *x* на *y*.

* Вставка вузла

1. Спочатку відбувається звичайна вставка як в бінарне дерево пошуку.
2. Вузол розфарбовується червоним.
3. Викликається допоміжна процедура, що відновлює червоно-чорні властивості (рух знизу вгору).

Можливі випадки:

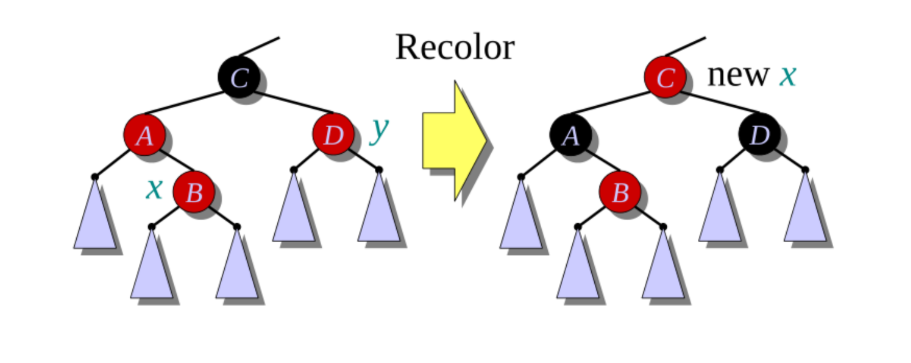
Прості випадки:

• новий вузол є коренем – перефарбовується в чорний;

• батьківський вузол чорний – властивості не порушені.

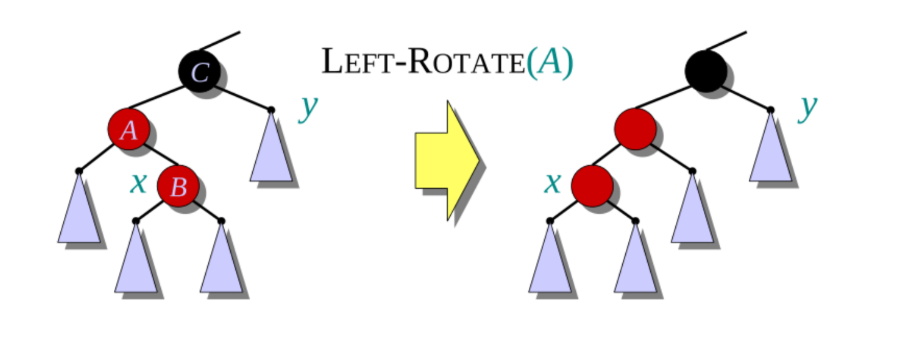
Інші випадки:

*Випадок 1.* «Дідусь» чорний, «дядько» y червоний.



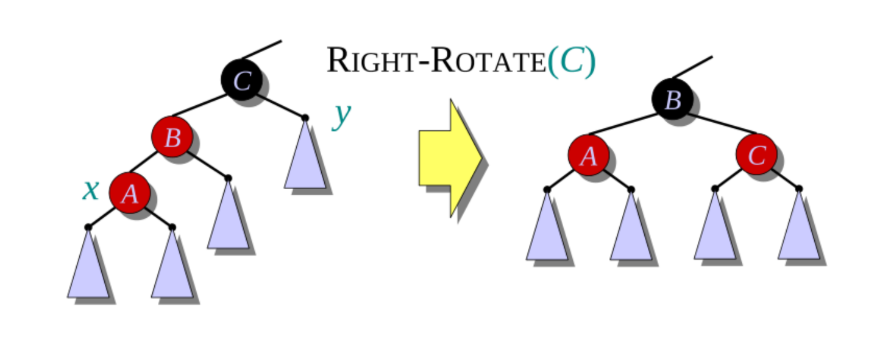
«Батько» і «дядько» стають чорними, «дідусь» червоним. Перевірка продовжується, бо предок С може виявитися червоним.

*Випадок 2.* «Дядько» чорний, х є правим потомком.



Лівий поворот відносно А. Зведення до випадку 3.

*Випадок 3.* «Дядько» чорний, х є лівим потомком.



Правий поворот відносно С та зміна кольорів. Вихід з алгоритму.

* Видалення вузла

Можливі три випадки залежно від наявності потомків:

• Якщо у вершини немає дітей, то змінюється вказівник на неї у батька на NIL.

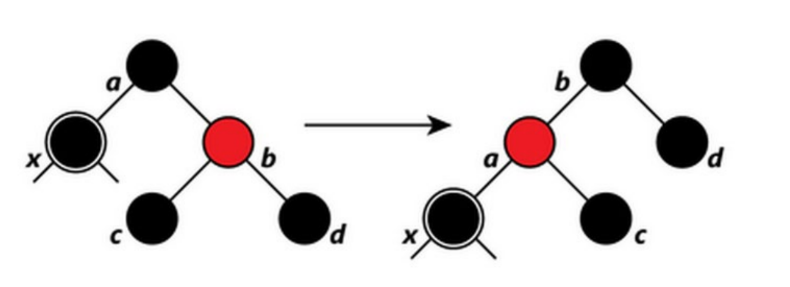
• Якщо у неї тільки один потомок, то у батька робиться посилання на нього замість цієї вершини.

• Якщо ж є обидва потомки, то знаходимо вершину з наступним значенням ключа. У такої вершини немає лівого дитини. Видаляємо вже цю вершину описаним вище способом, скопіювавши її ключ в початкову.

При видаленні червоної вершини червоно-чорні властивості не порушуються, тому перебалансування проводиться при видаленні чорного вузла. В процесі видалення може бути не більше 3 обертань.

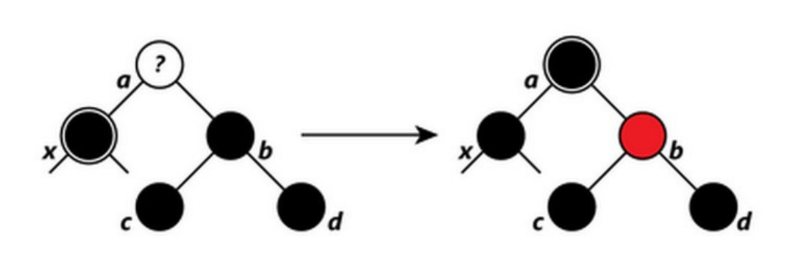
Можливі випадки:

*Випадок 1.* «Брат» вершини x червоний.



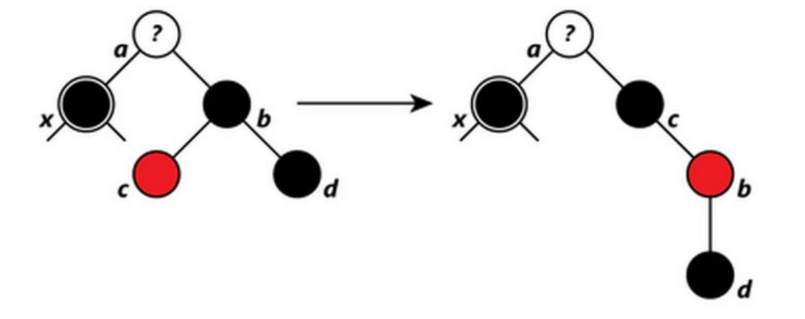
Ліве обертання відносно a та перефарбування. Зведення до випадків 2, 3, 4.

*Випадок 2.* «Брат» вершини x чорний. Обидва потомки «брата» чорні.



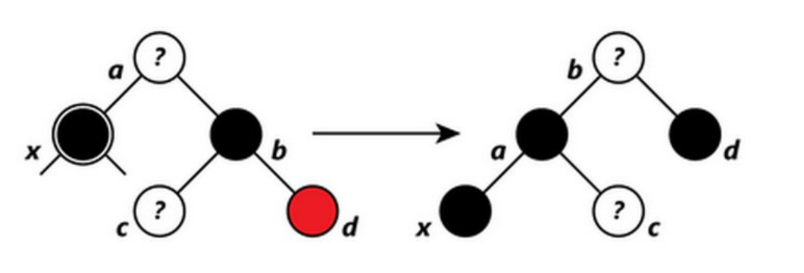
Перефарбовуємо «брата». Далі розглядаємо вузла-батька.

*Випадок 3.* «Брат» вершини x чорний. Лівий потомок «брата» червоний, правий чорний.



Перефарбовуємо «брата» і його лівого сина. Правий поворот відносно b. Зведення до випадку 4.

*Випадок 4.* «Брат» вершини x чорний. Правий потомок «брата» червоний.



Перефарбовуємо і робимо лівий поворот відносно a. Вихід з алгоритму.

**Складність.**

Всі операції для створення червоно-чорного дерева виконуються за час O(lg n).

**Мова реалізації.**

С++

**Модулі програми.**

* **Клас Rational.**

*Поля класу*: int num, den – чисельник та знаменник числа відповідно.

*Основні методи класу:*

Rational(int n = 0, int d = 1)– конструктор.

bool operator<(const Rational &other) const, bool operator>(const Rational &other) const, bool operator==(const Rational &other) const, Rational operator+(const Rational &other) const – деякі перевиражені оператори

friend std::ostream &operator<<(std::ostream &os, const Rational &r) – перевиражений оператор виводу.

* **Клас SegmentTreeRBT**

void leftRotate(SegmentNode \*x) – лівий поворот

void rightRotate(SegmentNode \*y) – правий поворот

void fixInsert(SegmentNode \*z) – відновлює властивості ЧЧД після вставки вузла

void fixDelete(SegmentNode \*x) – відновлює властивості ЧЧД після видалення вузла

void insert(const Rational &l, const Rational &r, const Rational &val) – вставка вузла

void remove(const Rational &l, const Rational &r) – видалення вузла

Rational querySum(const Rational &l, const Rational &r) – знаходження суми на інтервалі

Rational queryMin(const Rational &l, const Rational &r) - знаходження мінімального значення на інтервалі

Rational queryMax(const Rational &l, const Rational &r) - знаходження максимального значення на інтервалі

**Інтерфейс користувача.**

Введення даних може відбуватись як з програми, так і користувачем.

*Вхідні дані:*

інтервали і значення вхідинх вузлів

додатково за потреби: інтервали суму/мін/макс значення яких хочете знайти

*Вихідні дані:*

дерево порівнево (інтервал, значення, колір)

додатково за потреби: сума/мін/макс значення вказаного інтервалу

**Тестові приклади.**

**Приклад**

Приклад побудови дерева:

елементи: [3, 4] 5/7; [1/2, 3/2] 3/2; [5, 6] 2; [17/3, 8/1] 7/9.

Виконуючи коректну вставку всіх елементів у кінці отримаємо таке дерево:

[3, 4] 5/7

/ \

[1/2, 3/2] 3/2 [5, 6] 2

\

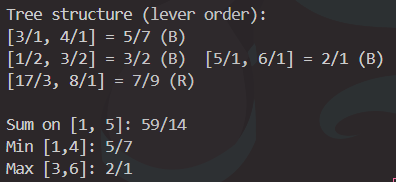
[17/3, 8/1] 7/9

Сума значень на відрізку [1, 5] = 3/2+5/7+2=59/14.

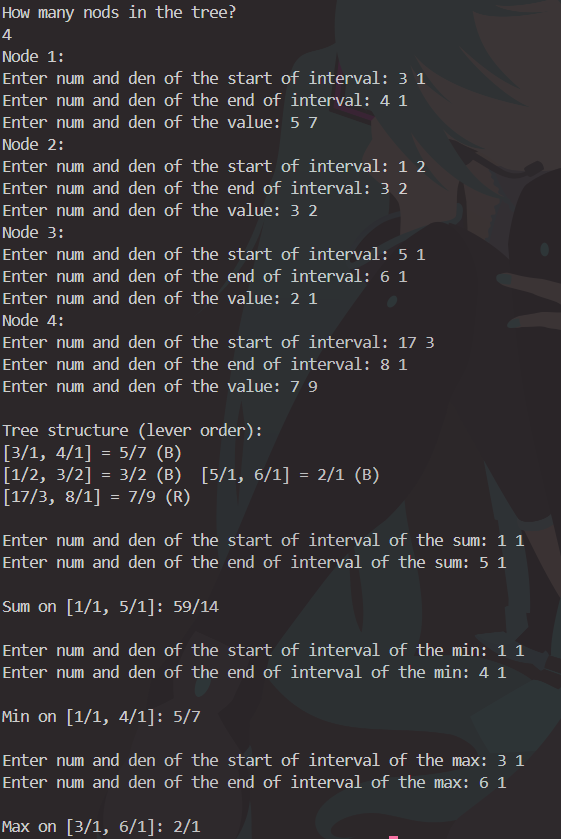
Мінімум на відрізку [1, 4] = 5/7.

Максимум на відрізку [3, 6] = 2.

*Ввід даних з програми:*



*Ввід даних користувачем:*



**Висновки.**

У ході виконання лабораторної роботи було реалізовано дерево відрізків на основі червоно-чорного дерева для раціональних чисел. Ми впевнелись, що у програмі коректно працюють функції додавання, видалення вузла, а також є можливість на певному відрізку знайти суму елементів, мінімальне або максимальне значення.

Використання червоно-чорного дерева гарантує логарифмічну складність основних операцій, зберігаючи баланс дерева після змін. Завдяки цьому структура залишається ефективною навіть при великій кількості елементів.

**Література.**

* Лекція з предмету «Алгоритми та складність #2»
* [**https://medium.com/@florian\_algo/segment-tree-1-basic-concepts-and-operations-in-detail-19a38c69648e**](https://medium.com/@florian_algo/segment-tree-1-basic-concepts-and-operations-in-detail-19a38c69648e)
* [**https://www.geeksforgeeks.org/introduction-to-red-black-tree/#4-rotation**](https://www.geeksforgeeks.org/introduction-to-red-black-tree/#4-rotation)
* [**https://uk.wikipedia.org/wiki/Червоно-чорне\_дерево**](https://uk.wikipedia.org/wiki/Червоно-чорне_дерево)