Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

**Лабораторна робота №3**

**Завдання: дерево порядкової статистики**

**(на основі червоно-чорного дерева)**

**Тип даних – Т9 (Комплексні числа)**

Виконав:

студент групи ІПС-23

факультету комп’ютерних наук

та кібернетики

Євчик Олексій

Київ 2023

**Зміст**

|  |  |
| --- | --- |
| Теоретичні відомості | 3 |
| Алгоритм | 4 |
| Складність | 5 |
| Мова програмування | 5 |
| Модулі програми | 6 |
| Інтерфейс користувача | 8 |
| Висновок | 9 |
| Джерела | 10 |
|  |  |

Теоретичні відомості

Дерево порядкової статистики є особливим видом бінарного дерева пошуку, яке зберігає додаткову інформацію про порядок елементів у дереві. Кожен вузол дерева зберігає значення інформації, яка може бути довільного типу даних, а також кількість вузлів в піддереві, що має цей вузол як корінь.

Однією з основних операцій, які можна виконати над деревом порядкової статистики, є отримання k-го елемента у відсортованому списку всіх елементів, що зберігаються у дереві. Ця операція називається пошуком k-ої порядкової статистики.

Для досягнення цієї мети, дерево порядкової статистики зберігає додаткову інформацію у вузлах дерева, яка дозволяє швидко знаходити k-ий елемент. Зокрема, кожен вузол містить кількість вузлів у піддереві, який має лівого нащадка. Ця інформація використовується для того, щоб визначити, в якому напрямку здійснювати пошук k-ої порядкової статистики: у лівому піддереві, правому піддереві, або в поточному вузлі.

В даній лабораторній роботі я використовував як основу червоно-чорне дерево.

**Алгоритм**

Основний ідея алгоритму порядкової статистики на основі червоно-чорного дерева полягає в тому, щоб підрахувати кількість вузлів у піддереві кожного вузла, які є меншими за цей вузол. Це дозволяє швидко знайти к-тий елемент у дереві.

Алгоритм для дерева порядкової статистики містить наступні операції:

1. Вставка(Insert): Додавання нового елемента у дерево відбувається шляхом вставки елемента до звичайного бінарного дерева пошуку, а потім фіксації дерева, щоб забезпечити, що воно залишається червоно-чорним деревом. Для фіксації дерева використовуються різні методи, такі як обертання вправо та вліво.
2. Пошук к-тої статистики(OrderStatistic): Знаходження k-ого елементу у дереві порядкової статистики виконується шляхом обходу дерева вгору та/або вниз, знаходячи k-ий елемент. Для цього здійснюється порівняння k з розміром лівого піддерева поточного вузла x. Якщо k менше розміру лівого піддерева x, то шуканий елемент знаходиться в лівому піддереві x. Якщо k дорівнює розміру лівого піддерева x + 1, то x є шуканим елементом. Якщо k більше розміру лівого піддерева x, то шуканий елемент знаходиться у правому піддереві x. Для кожного вузла x під час обходу дерева вгору та/або вниз порівнюється k з розміром лівого піддерева x, що дозволяє знаходити шуканий елемент у дереві за O(log n) операцій.
3. Виправлення(FixTreeAfterInsert): Цей метод використовується для фіксування дерева після вставки нового вузла, щоб забезпечити, що воно залишається червоно-чорним деревом.

Алгоритм додавання нового елемента до дерева порядкової статистики:

1. Вставити новий вузол в дерево, як в звичайному червоно-чорному дереві.
2. Змінити розмір (size) всіх вузлів на шляху від нового вузла до кореня.
3. Викликати метод FixTreeAfterInsert(), який відновлює властивості червоно-чорного дерева.

Алгоритм визначення k-го елемента:

1. Почати пошук з кореня дерева.
2. Якщо розмір (size) лівого піддерева дорівнює k-1, то поточний вузол є k-м елементом.
3. Якщо розмір (size) лівого піддерева менший за k-1, то рекурсивно шукати k-й елемент в правому піддереві.
4. Якщо розмір (size) лівого піддерева більший за k-1, то рекурсивно шукати (k-1)-й елемент в лівому піддереві.

Ці алгоритми забезпечують швидкий доступ до k-го елементу дерева порядкової статистики.

Алгоритм виправлення можна описати наступним чином:

1. Перевірте, чи вузол є коренем дерева. Якщо так, то присвойте йому чорний колір.
2. Якщо батьківський вузол нового вузла є чорним, то немає необхідності виконувати додаткові операції.
3. Якщо батьківський вузол нового вузла є червоним, то перевірте, чи дядько нового вузла (брат батька) є червоним.
4. Якщо дядько нового вузла є червоним, то змініть колір батька та дядька на чорний, а колір дідуся (батько батька) на червоний.
5. Якщо дядько нового вузла є чорним, то виконайте обертання вправо чи вліво, щоб забезпечити балансування дерева. Залежно від структури дерева може знадобитися одне чи кілька обертань. Після цього змініть колір дідуся на червоний та колір батька на чорний.
6. Перевірте, чи дерево залишилося коректним червоно-чорним деревом. Якщо ні, то повторіть кроки 2-5 для батька нового вузла та його дідуся.

**Складність**

Вставка елемента: складність вставки в дерево порядкової статистики на основі червоно-чорного дерева є O(log n), де n - кількість елементів у дереві. Це завдяки тому, що червоно-чорне дерево гарантує балансування дерева, що забезпечує глибину дерева не більше O(log n).

Фіксація дерева після вставки: фіксація дерева може знадобитися після кожної вставки, щоб забезпечити, що дерево залишається червоно-чорним. Складність фіксації дерева також є O(log n), оскільки максимальна глибина дерева дорівнює O(log n).

Пошук порядкової статистики: складність пошуку k-го елемента в порядку зростання є O(log n), де n - кількість елементів у дереві. Це завдяки тому, що ми обходимо дерево вгору та/або вниз, поки не знайдемо k-ий елемент.

Мова програмування

Мова програмування: С++;

Середовище розробки: Visual Studio 2022

GitHub: <https://github.com/MoloZzz/lab3AaD>

Модулі(документація)

**Опис**

Клас **OrderStatisticTree** - це реалізація дерева порядку статистики на основі червоно-чорного дерева.

Це дерево, яке зберігається у вигляді червоно-чорного дерева, що містить комплексні числа **Complex**. Кожен вузол містить додаткову інформацію про розмір піддерева, що є коренем цього вузла.

Клас **OrderStatisticTree** має дві основні операції:

* **Insert(Complex data)**: Додає новий вузол з даними **data** до дерева.
* **OrderStatistic(int k)**: Повертає **k**-те за порядком статистики комплексне число з дерева. Наприклад, якщо **k = 1**, повертається найменше число в дереві. Якщо **k = n**, повертається найбільше число в дереві, де **n** - загальна кількість елементів в дереві.

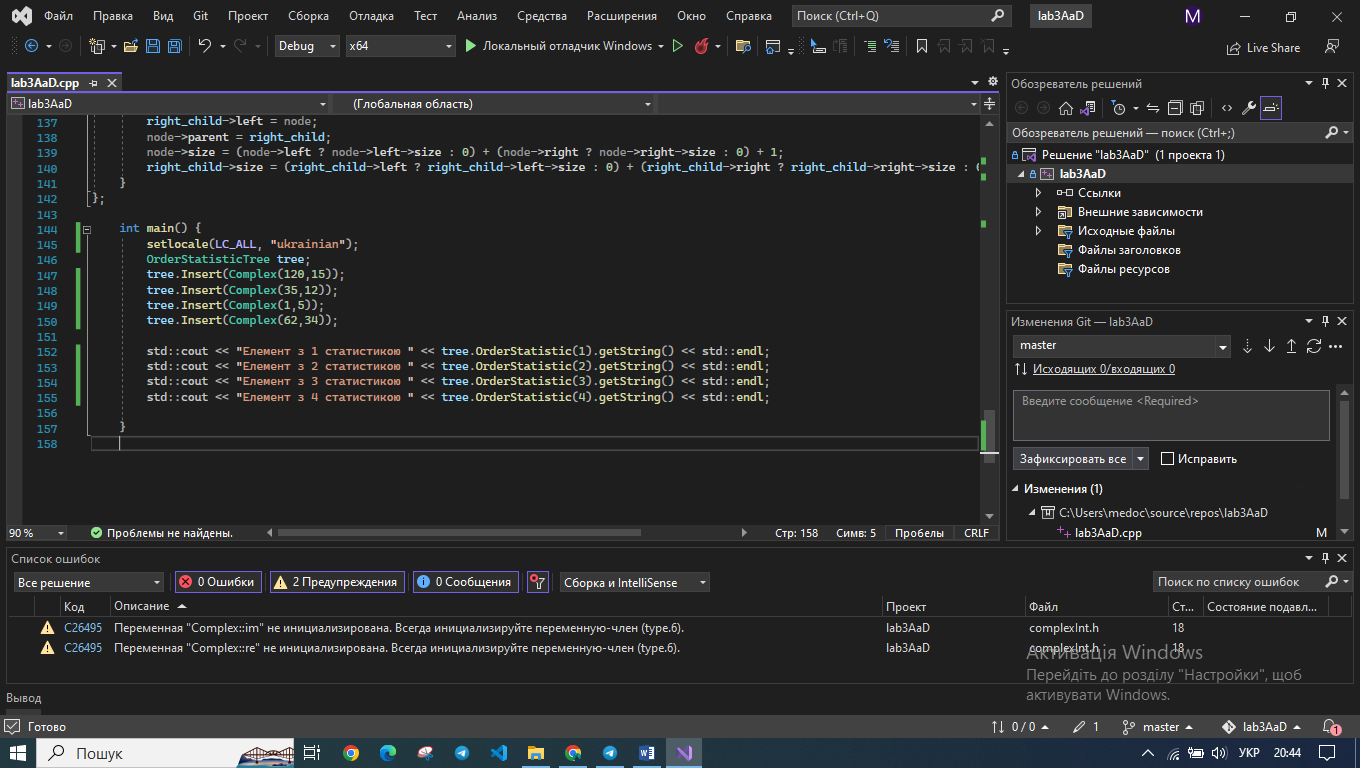
**Поля**

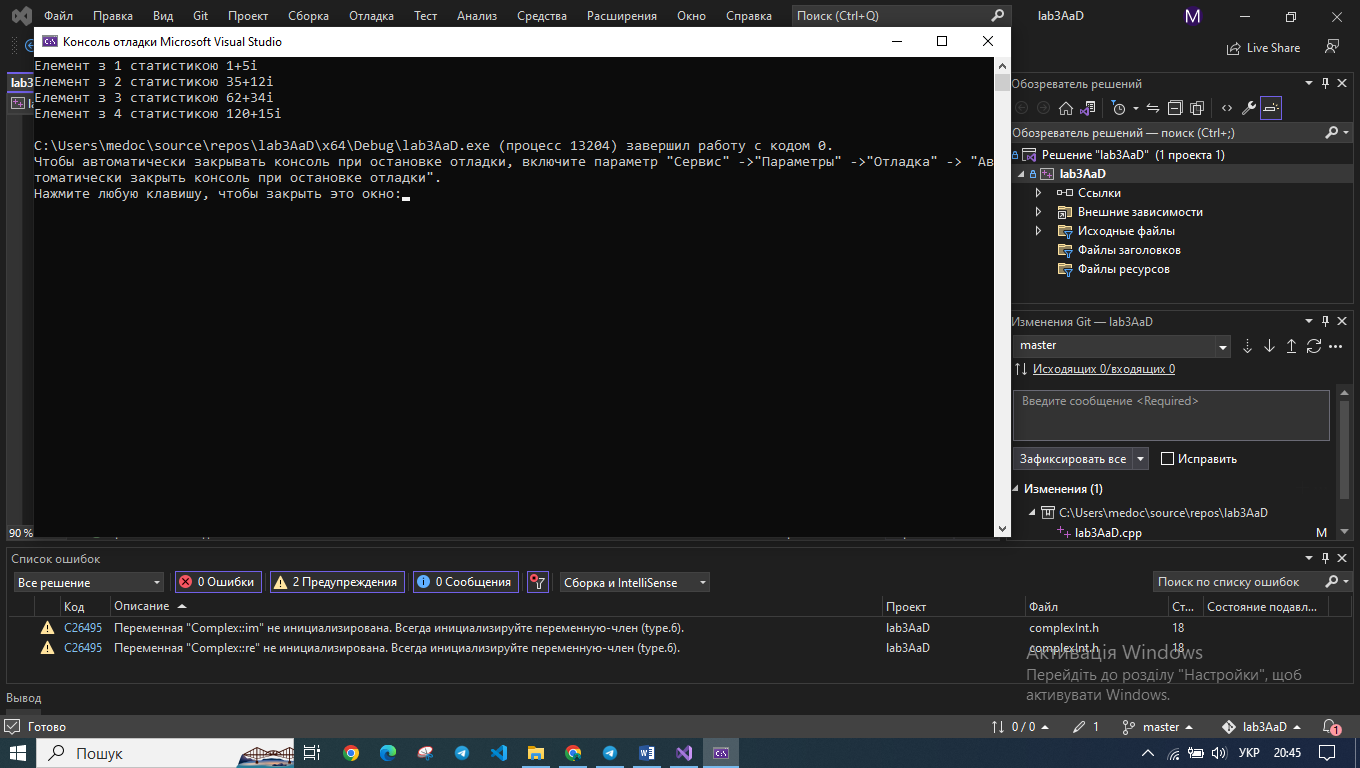
* **root**: вказівник на корінь дерева.

**Методи**

* **OrderStatisticTree()**: конструктор за замовчуванням, який ініціалізує поле **root** вказівником на **nullptr**.
* **Insert(Complex data)**: додає новий вузол з даними **data** до дерева.
* **OrderStatistic(int k)**: повертає **k**-те за порядком статистики комплексне число з дерева.
* **FixTreeAfterInsert(Node\* node)**: коригує дерево після вставки нового вузла з даними **node**.
* **RotateRight(Node\* node)**: виконує правий поворот дерева вздовж зазначеного вузла **node**.
* **RotateLeft(Node\* node)**: виконує лівий поворот дерева вздовж зазначеного вузла **node**.

**Інтерфейс користувача**





**Висновок**

Загалом, дерева порядкової статистики є корисною структурою даних, яка дозволяє швидко знаходити k-ий елемент у відсортованій послідовності за O(log n) операцій. Ця структура даних є підмножиною бінарного дерева пошуку, але містить додаткову інформацію про кількість елементів у кожному піддереві, що дозволяє виконувати операцію пошуку k-го елементу ефективно.

Алгоритм для дерева порядкової статистики включає операції вставки, пошуку k-го елементу та фіксації дерева після вставки нового елементу. Для фіксації дерева після вставки використовуються правила червоно-чорного дерева, що забезпечує балансування дерева та оптимальну швидкість виконання операцій.

У загальному, дерева порядкової статистики є корисним інструментом для розв'язання різноманітних задач, пов'язаних з пошуком k-го елементу у відсортованій послідовності.

**Джерела**

<https://www.geeksforgeeks.org/order-statistic-tree-using-fenwick-tree-bit/>

<https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%88%D1%83%D0%BA_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%BE%D0%B2%D0%BE%D1%97_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B8>

<https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D0%BE%D0%B8%D1%81%D0%BA_k-%D0%BE%D0%B9_%D0%BF%D0%BE%D1%80%D1%8F%D0%B4%D0%BA%D0%BE%D0%B2%D0%BE%D0%B9_%D1%81%D1%82%D0%B0%D1%82%D0%B8%D1%81%D1%82%D0%B8%D0%BA%D0%B8>

<https://en.wikipedia.org/wiki/Order_statistic_tree>

<https://www.cs.yale.edu/homes/aspnes/pinewiki/OrderStatisticsTree.html>

<http://www.cs.cornell.edu/courses/cs211/2004su/slides/Topic20b.pdf>