Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Лабораторна робота №2

“Реалізація декартового дерева.”

Виконав студент 2-го курсу

Групи ІПС-22

Каширець Роман Віталійович

2025

**Завдання**

Реалізуємо декартове дерево (Treap) для зберігання комплексних чисел з цілочисельними компонентами, де порядок визначається спочатку за модулями чисел, а для чисел з однаковими модулями - за першою (дійсною) компонентою.

**Теорія**

Декартове дерево (Treap) - це гібридна структура даних, яка поєднує властивості **бінарного дерева пошуку** (BST) та **двійкової купи** (heap). Назва "Treap" походить від слів "tree" (дерево) і "heap" (купа).

Властивості Treap:

1. **BST-властивість**: Для кожного вузла X:
   * Усі вузли в лівому піддереві мають ключі, менші за ключ X
   * Усі вузли в правому піддереві мають ключі, більші за ключ X
2. **Heap-властивість**: Для кожного вузла X:
   * Пріоритет X більший або рівний пріоритетам його дочірніх вузлів

Переваги:

* Середня складність операцій O(log n)
* Не вимагає складних алгоритмів балансування (як AVL або червоно-чорні дерева)
* Простота реалізації
* Ефективна підтримка операцій об'єднання та розділення дерев

**Алгоритм**

**1. Алгоритм вставки**

**Кроки:**

1. Починаємо з кореня дерева.
2. Порівнюємо значення нового комплексного числа з поточним вузлом:
   * Спочатку порівнюємо модулі чисел
   * При рівних модулях - порівнюємо дійсні частини
3. Якщо значення менше:
   * Рекурсивно вставляємо в ліве піддерево
   * Після вставки перевіряємо пріоритети:
     + Якщо пріоритет лівого дочірнього вузла вищий - виконуємо праву ротацію
4. Якщо значення більше або рівне:
   * Рекурсивно вставляємо в праве піддерево
   * Після вставки перевіряємо пріоритети:
     + Якщо пріоритет правого дочірнього вузла вищий - виконуємо ліву ротацію
5. Повертаємо оновлений вузол.

Особливості для комплексних чисел:

* Порівняння відбувається за модулем (√(a² + b²))
* При однакових модулях використовуємо дійсну частину (Re)

**2. Алгоритм видалення**

**Кроки:**

1. Знаходимо вузол, який потрібно видалити, використовуючи BST-властивість.
2. Якщо вузол - листок (немає дітей), просто видаляємо його.
3. Якщо вузол має тільки одного нащадка, замінюємо його цим нащадком.
4. Якщо вузол має двох нащадків:
   * Порівнюємо їх пріоритети
   * Виконуємо ротацію в бік вузла з вищим пріоритетом
   * Продовжуємо рекурсивно видаляти вузол у відповідному піддереві
5. Повертаємо оновлений вузол.

**Важливо: При видаленні ми спочатку опускаємо вузол до рівня листка за допомогою ротацій, перш ніж видалити його.**

**3. Алгоритм пошуку**

Кроки:

1. Починаємо з кореня дерева.
2. Порівнюємо шукане значення з поточним вузлом:
   * Спочатку за модулем
   * Потім за дійсною частиною (якщо модулі рівні)
3. Якщо значення збігаються - повертаємо вузол.
4. Якщо шукане значення менше - переходимо до лівого піддерева.
5. Якщо шукане значення більше - переходимо до правого піддерева.
6. Якщо досягли кінця дерева (null) - елемент не знайдено.

**4. Алгоритми ротацій**

Права ротація:

1. Беремо лівий дочірній вузол поточного вузла (назвемо його X).
2. Переміщуємо праве піддерево X у ліве піддерево поточного вузла.
3. Робимо X новим коренем піддерева.
4. Повертаємо X як новий корінь.

Ліва ротація:

1. Беремо правий дочірній вузол поточного вузла (назвемо його Y).
2. Переміщуємо ліве піддерево Y у праве піддерево поточного вузла.
3. Робимо Y новим коренем піддерева.
4. Повертаємо Y як новий корінь.

**5. Алгоритм обходу (in-order traversal)**

Кроки:

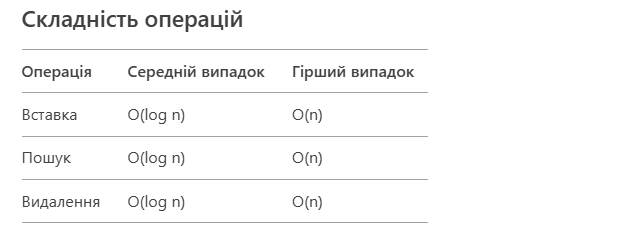
1. Починаємо з кореня дерева.
2. Рекурсивно обходимо ліве піддерево.
3. Виводимо значення поточного вузла.
4. Рекурсивно обходимо праве піддерево.

Результат: Відсортована послідовність комплексних чисел за зростанням (спочатку за модулем, потім за дійсною частиною).

**Особливості для комплексних чисел:**

1. Ключі порівнюються за модулем: |z| = √(Re² + Im²)
2. При однакових модулях використовуємо лексикографічне порівняння:
   * Спочатку Re (дійсна частина)
   * Потім Im (уявна частина), якщо Re однакові
3. Пріоритети генеруються випадково при створенні вузла

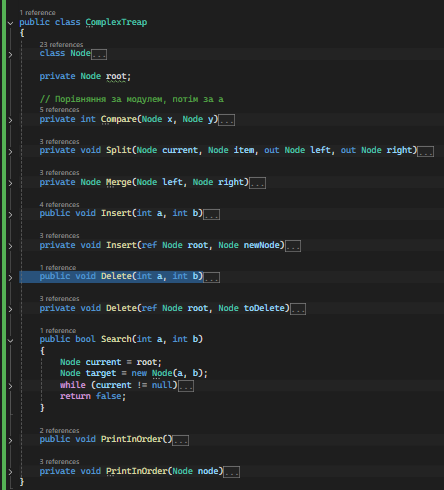
**Складність**



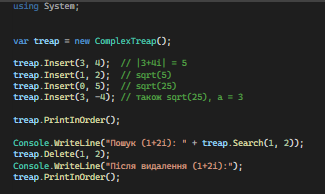
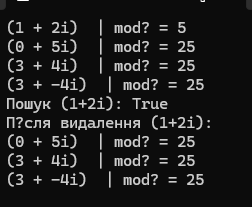
**Мова програмування**

**С#**

**Модулі програми**

****

**Інтерфейс користувача**

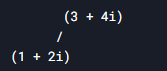
  
**Алгоритм**Алгоритм роботи ComplexTreap

КРОК 1: Вставка (3 + 4i)

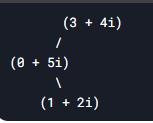
1. **Створення вузла**:
   * Дані: a=3, b=4
   * Ключ: 3² + 4² = 25
   * Пріоритет: випадкове число
2. **Вставка**:
   * Дерево порожнє → вузол стає коренем 

КРОК 2: Вставка (1 + 2i)

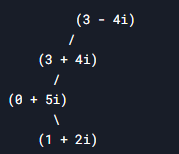
1. **Створення вузла**:
   * Дані: a=1, b=2
   * Ключ: 1² + 2² = 5
   * Пріоритет: випадкове число
2. **Порівняння з коренем (3+4i)**:
   * 5 < 25 → вставляємо ліворуч
   * Пріоритет нового вузла менший за пріоритет кореня → проста вставка

  
  
КРОК 3: Вставка (0 + 5i)

1. **Створення вузла**:
   * Дані: a=0, b=5
   * Ключ: 0² + 5² = 25
   * Пріоритет: випадкове число
2. **Порівняння**:
   * Ключі рівні (25 == 25) → порівнюємо a: 0 < 3 → вставляємо ліворуч
   * Проходимо до (1+2i):
     + 25 > 5 → вставляємо праворуч від (1+2i)
3. **Балансування**:
   * Якщо пріоритет нового вузла вищий за (1+2i) → виконуємо Split і Merge

  
КРОК 4: Вставка (3 - 4i)

1. **Створення вузла**:
   * Дані: a=3, b=-4
   * Ключ: 3² + (-4)² = 25
   * Пріоритет: випадкове число
2. **Порівняння з коренем (3+4i)**:
   * Ключі рівні (25 == 25), a рівні (3 == 3) → вставляємо праворуч (умовно)
3. **Балансування**:
   * Якщо пріоритет вищий за корінь → Split дерева, новий вузол стає коренем



**КРОК 5: Пошук (1 + 2i)**

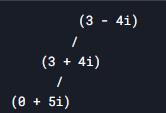
1. Алгоритм:
   * Ключ пошуку: 1² + 2² = 5
   * Порівнюємо з коренем (3-4i, ключ 25):
     + 5 < 25 → йдемо ліворуч до (3+4i)
   * Порівнюємо з (3+4i):
     + 5 < 25 → йдемо ліворуч до (0+5i)
   * Порівнюємо з (0+5i):
     + 5 < 25 → йдемо ліворуч (немає)
     + Праворуч знаходимо (1+2i) → знайдено

Результат: True

**КРОК 6: Видалення (1 + 2i)**

1. Знаходимо вузол (аналогічно пошуку)
2. Видаляємо:
   * Замінюємо його Merge його піддерев
   * Якщо вузол - листок, просто видаляємо

Стан дерева після видалення:

**  
**

**Висновок**

Реалізований **Treap для комплексних чисел** є елегантним рішенням для динамічних наборів даних із багаторівневим порядком. Він поєднує простоту реалізації з ефективністю, що робить його відмінним вибором для завдань, де потрібна швидка робота зі складними ключами.

**Використана література**

** G​eeksforGeeks – Implementation of Search, Insert and Delete in Treap\*\***Детальний опис реалізації операцій пошуку, вставки та видалення в декартовому дереві, з прикладами коду.

**\*\*C​odeProject–TreapsinC#\*\***  
Стаття з прикладом реалізації Treap на C#, включаючи пояснення принципів роботи та коду.

 **\*\*M​edium – A Visual Introduction to Treap Data Structure (Part I: The Basics)\*\***Візуальне пояснення основ Treap, включаючи принципи роботи та переваги цієї структури даних.