**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

ЛАБОРАТОРНА РОБОТА №2

З курсу “Алгоритмі і складність”

Декартове дерево

Виконав

Студент групи ПІ-22

Факультету комп’ютерних наук

та кібернетики

Мандзюк Дмитро

Київ 2023

**Зміст**

[Завдання 3](#_Toc128161844)

[Теорія 3](#_Toc128161845)

[Алгоритм 3](#_Toc128161846)

[Складність 7](#_Toc128161847)

[Мова програмування 7](#_Toc128161848)

[Модулі програми 7](#_Toc128161849)

[Інтерфейс користувача 8](#_Toc128161850)

[Приклади 8](#_Toc128161851)

[Висновок 9](#_Toc128161852)

[Література 10](#_Toc128161853)

# **Завдання**

Реалізувати декартове дерево для **раціональних чисел**.

# **Теорія**

**Декартове дерево** – структура, що поєднує в собі бінарне дерево пошуку і купу. Кожен вузол містить значення ключа та пріоритету. Тобто структура є **деревом пошуку** за ключами та **купою** (**пірамідою**) за пріоритетами. Ключі можуть повторюватися, але дублі мають знаходитися однозначно тільки справа чи зліва. (Пріоритетом є випадкове число з заданого проміжку

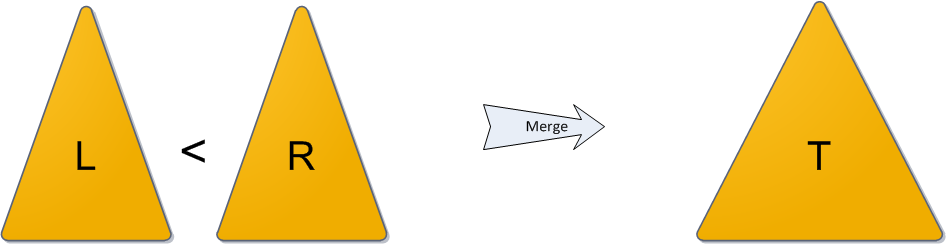
(наприклад, (0, 1). Повторів пріоритетів слід уникати).

В даному випадку в якості обмеження для пріоритету був обраний проміжок від 0 до 1.

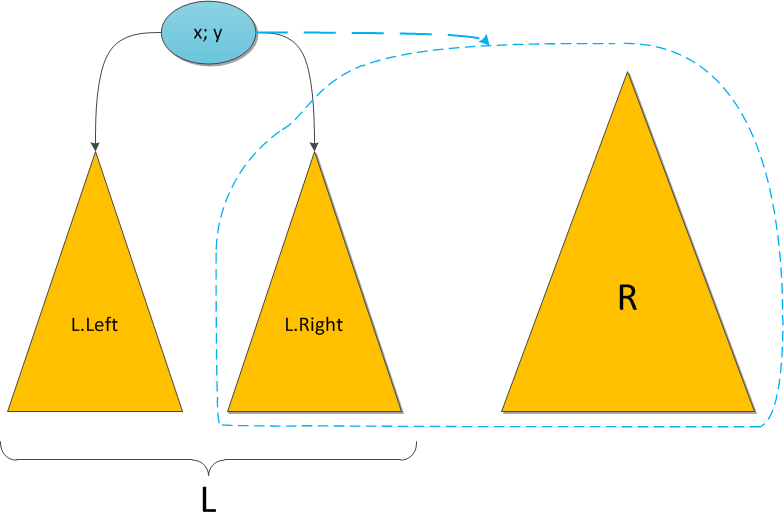
# **Алгоритм**

Для декартового дерева визначаємо дві базові операції: **Merge** і **Split**.

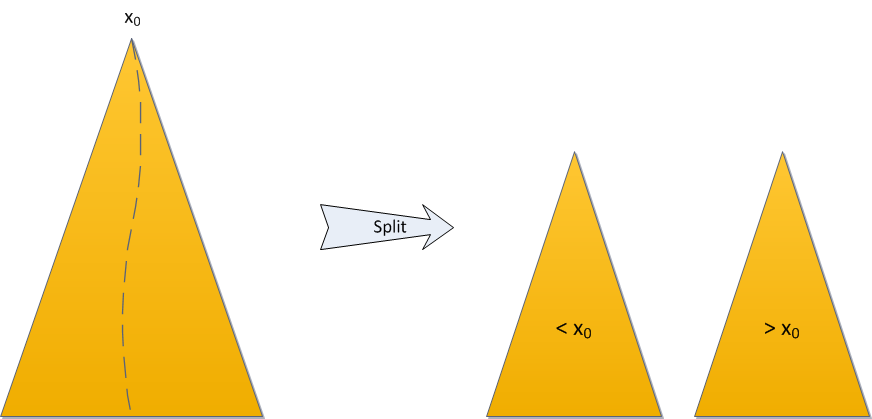
1. Операція Merge приймає на вхід два декартові дерева L і R. Вона зливає їх в одне, теж коректне, декартове дерево T.

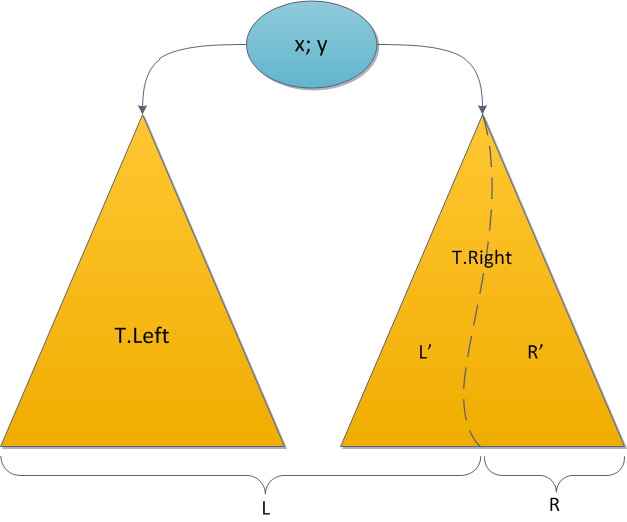


Алгоритм Merge досить простимй. В якості нового кореня береться корінь дерева з більшим пріоритетом. Одне з піддерев очевидне, друге отримується рекурсивно. Пріоритет лівого кореня більший.



1. Операція Split приймає на вхід ключ x і декартове дерево. Вона розбиває наявне декартове дерево на два дерева L і R, де в дереві L всі ключі не перевищую x, а в дереві R всі ключі більші за x.

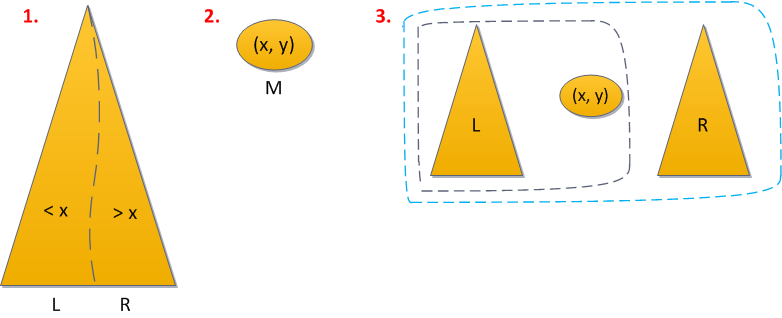
Звіряємо корінь з ключем. В залежності від результату бачимо, до якого з нових дерев належатиме корінь з одним із піддерев. Друге дерево рекурсивно виділяється з іншого піддерева.



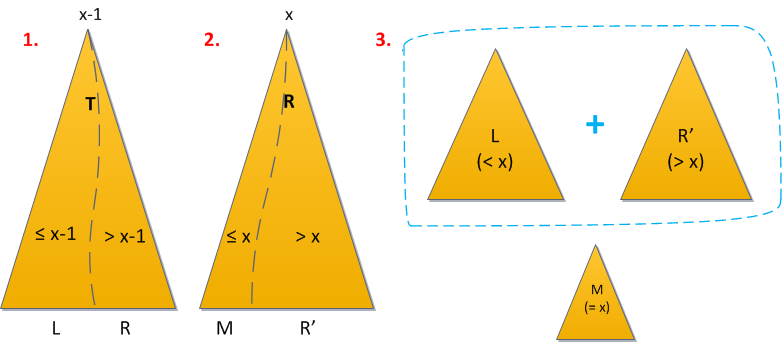
1. Операція **Search** - пошук елемента в дереві за ключем x. Повертає вказівник на відповідний вузол.

Те саме, що й в звичайному бінарному дереві пошуку.

1. Операція **Insert** – вставка x в дерево.
2. Розділимо (**split**) дерево за ключом x на дерево L, з ключами менше x, і дерево R, з більшими.
3. Створимо з цього ключа дерево M з єдиної вершини (x, y), де y - щойно згенерований випадковий пріоритет.
4. Об'єднаємо (**merge**) по черзі L з M, а що вийшло - з R.



1. Операція Delete – видалення за ключем.
2. Розділимо спочатку дерево за ключом x-1/1000. Усі елементи, менші чи рівні x-1/1000, вирушили у лівий результат, отже, шуканий елемент — у правому.
3. Розділимо правий результат за ключом x. У новий правий результат вирушили всі елементи з ключами, більшими x, а «середній» (лівий від правого) — менші чи рівні x. Але оскільки всі ключі строго менші за x після першого кроку були відсіяні, то середнє дерево і є елементом, що шукається.
4. Тепер поєднаємо знову ліве дерево з правим, без середнього, і дерево залишиться без ключів x.



# **Складність**

Складність створення дерева у найгіршому випадку (через метод, який ми використовуємо тут) - O(n ×log n).

Складність додавання елементу до дерева в середньому - O(log n).

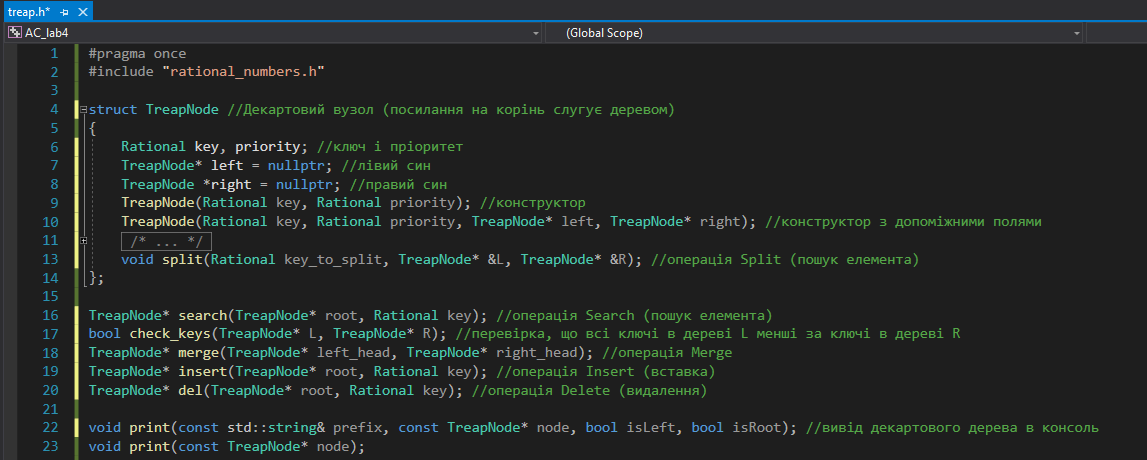
Складність пошуку елемента користувачем в середньому - O(log n).

Складність видалення елементу з дерева в середньому - O(log n).

# **Мова програмування**

С++

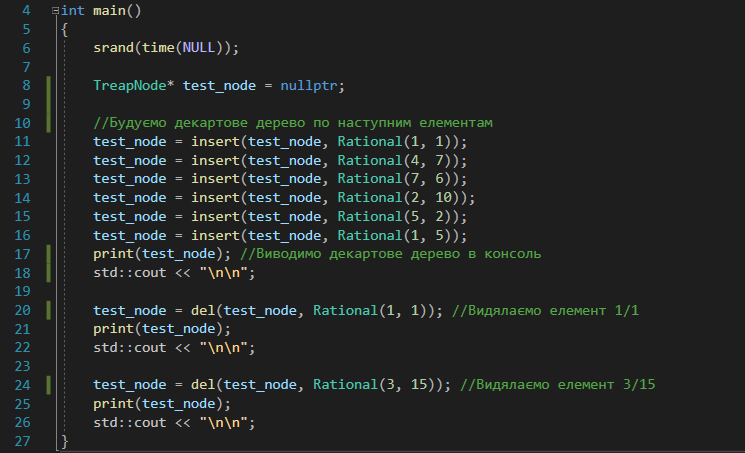
# **Модулі програми**

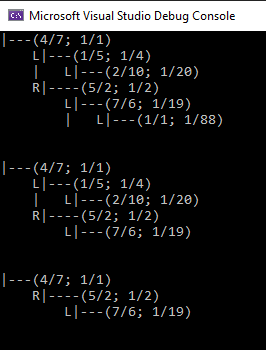


# **Інтерфейс користувача**

Вхідні данні задаються в коді програми (а саме в наведеній вище функції int(main)), а результат виводиться у консоль.

# **Приклади**





Програма успішно створила дерево по заданим точкам, вивила його в консоль і видалила вказані елементи. Також варто звернути увагу, що коли була введена команда Delete для елемента 3/15 програма видалила всі наявні дублікати.

# **Висновок**

Декартове дерево має своє переваги і недоліки. Воно простіше реалізується в порівнянні, наприклад, зі справжніми деревами, що самобалансуються, наприклад червоно-чорним. Також воно добре поводиться «в середньому», якщо ключі роздати випадково: типова для сортуючого дерева операція «розділити за ключом x на „менше x0“ та „не менше x0“» (“**split**”) працює за O(h), де h — висота дерева. На червоно-чорних деревах довелось б відновлювати балансування та фарбувати вузли.  
Водночас до недоліків можна віднести великі витрати на зберігання: разом з кожним елементом зберігаються два-три покажчики та випадковий ключ y. Також в деяких ситуаціях декартове дерево може дуже сильно розбалансуватись і його висота буде рівною кількості елементів (O(n)). Тому, наприклад, декартове дерево не використовується у ядрах операційних систем.

# **Література**

1. Лекція з предмету «Алгоритми та складність 3»
2. <https://web.archive.org/web/20110605030306/http://www.cs.uiuc.edu/class/sp09/cs473/notes/08-treaps.pdf>
3. https://habr.com/ru/post/101818/