Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних інформацій систем

Алгоритми та складніcть

Лабораторна робота №2.2

Завдання №2.2 Реалізувати розширюване дерево (splay tree)

Тип даних – Раціональні числа

Виконала студентка 2-го курсу

Групи ІПС - 22

Мандріченко Ксенія

2023

**Зміст**

[Теоретичні відомості 3](#_Toc102339827)

[Алгоритм 3](#_Toc102339828)

[Складність 9](#_Toc102339829)

[Мова програмування 9](#_Toc102339830)

[Модулі програми 10](#_Toc102339831)

[Інтерфейс користувача 11](#_Toc102339832)

[Приклади 11](#_Toc102339833)

[Висновок 12](#_Toc102339834)

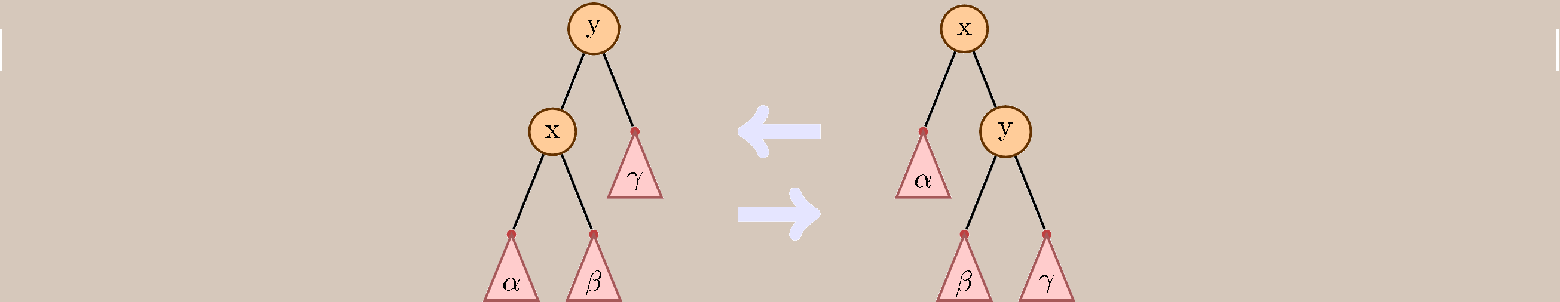
[Література 13](#_Toc102339835)

# **Теоретичні відомості**

**Сплей-дерево (Splay-tree)** - це двійкове дерево пошуку. Воно дозволяє знаходити швидше ті дані, які використовувалися нещодавно. Сплей-дерево було придумано Робертом Тар'яном та Данієлем Слейтером у 1983 році.

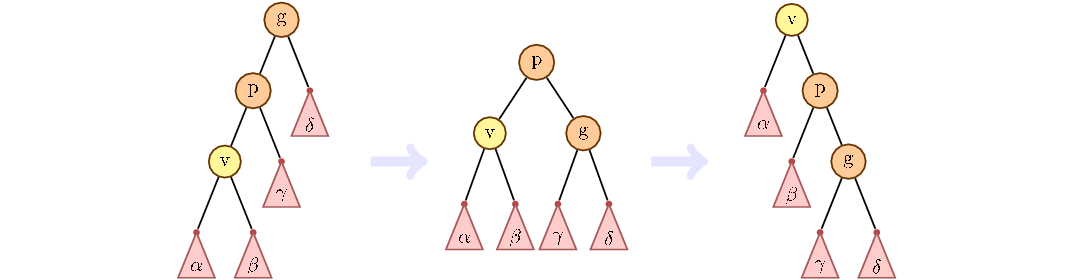
**Властивості Splay -дерева:**

Після кожного звернення, навіть пошуку, splay-дерево змінює свою структуру. Після звернення до будь-якої вершини вона піднімається в корінь. Підйом реалізується через повороти вершин. За один поворот можна поміняти місцями батька з дитиною, як показано на малюнку нижче.

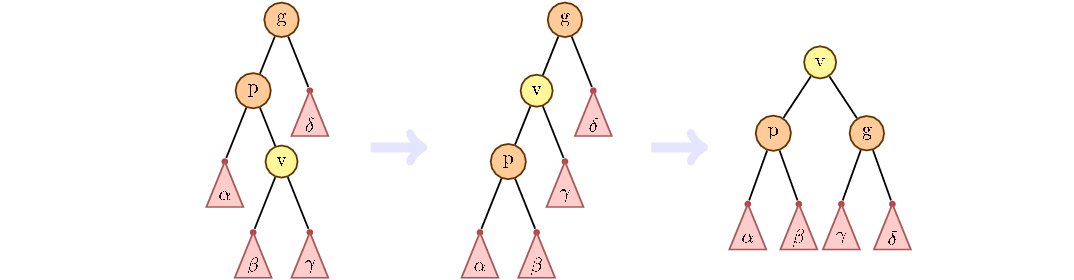


Але просто повертати вершину, доки вона не стане коренем, недостатньо. Хитрість splay-дерева в тому, що при просуванні вершини вгору, відстань до кореня скорочується не тільки для вершини, що піднімається, але і для всіх її нащадків у поточних піддеревах. Для цього використовується техніка zig-zig та zig-zag поворотів.

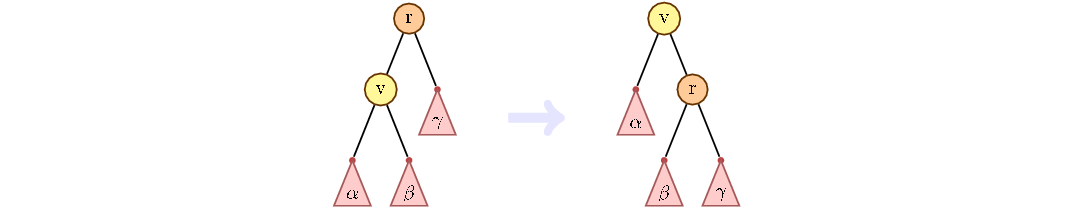
Основна ідея zig-zig та zig-zag поворотів, розглянути шлях від прабатька до дитини. Якщо шлях йде тільки лівими дітьми або тільки правими, то така ситуація називається zig-zig. Як її обробляти показано на малюнку нижче. Спочатку повернути батька, потім дитину.



В іншому випадку, ми спочатку змінюємо дитину з поточним батьком, потім з новим. Така ситуація називається zig-zag.



Якщо у вершини прабатька немає, робимо звичайний поворот:



* Процедура пошуку в splay-дереві відрізняється від звичайної лише на останній стадії: після того, як вершина знайдена, ми тягнемо її вгору і робимо коренем через процедуру splay.
* Щоб реалізувати вставку та видалення ключа, нам знадобляться дві процедури: split та merge (розрізати та злити).
* Процедура split отримує на вхід ключ key та ділить дерево на два. В одному дереві всі значення менші за ключ key, а в іншому — більше. Реалізується вона просто. Потрібно через find знайти найближчу до ключа вершину, витягнути її вгору і потім відрізати або ліве, або праве поддерево (або обидва).
* Процедура merge отримує на вхід два дерева: ліве left та праве right. Для коректної роботи, ключі дерева left повинні бути меншими за ключі дерева right. Тут ми беремо вершину з найбільшим ключем лівого дерева та тягнемо її догори. Після цього ,як праве піддерево, приєднуємо дерево right.
* Для того щоб видалити вершину, потрібно підняти її вгору, а потім злити її ліві та праві піддерева, та видалити вершину .

# **Алгоритм**

Програма визначає клас Rational, який представляє раціональні числа, і клас SplayTree, який реалізує самозбалансоване бінарне дерево пошуку .

Клас Rational має два приватних цілих члени даних, num і den, що представляють чисельник і знаменник раціонального числа.

Він також має приватний метод gcd, який обчислює найбільший спільний дільник двох цілих чисел, який використовується для зведення раціонального числа до його найпростішої форми. Зведення раціонального числа до його найпростішої форми полегшує виконання математичних операцій над раціональним числом і порівняння його з іншими раціональними числами.

Клас Rational має два публічні оператори порівняння (<, == ), які порівнюють два об’єкти Rational. У цій програмі два раціональні числа порівнюються шляхом перехресного множення їхніх знаменників і порівняння отриманих чисельників. Якщо ліва сторона менша за праву, функція повертає true, вказуючи, що ліве раціональне число менше за праве.

Клас SplayTree має приватну вкладену структуру Node, яка представляє вузол у Splay - дереві. Кожен вузол містить об’єкт Rational і покажчики на його батьківський, лівий і правий дочірні вузли.

Клас SplayTree має кілька приватних методів, які реалізують алгоритм splay-дерева, включаючи rotateLeft, rotateRight, splay, findNode, split і merge:

* Метод rotateLeft і rotateRight виконують обертання ліворуч і праворуч, відповідно, на даному вузлі для підтримки балансу дерева.
* Метод splay виконує послідовність обертань на заданому вузлі, щоб перемістити його до кореня дерева, зберігаючи порядок вузлів. Функція приймає параметр x, який є вузлом, який потрібно розширити до кореня. Цикл while неодноразово перевіряє, чи батьківський вузол поточного вузла дорівнює нулю (це означає, що поточний вузол є кореневим), і виконує послідовність обертань на основі положення вузла відносно його батьківського та прабатьківського вузлів.
  + Перший оператор if обробляє випадок, коли батько x є коренем дерева. Залежно від того, чи є x лівим чи правим дочірнім елементом свого батька, функція виконує один поворот, щоб перемістити x до кореня.
  + (zig-zig) випадки, коли x не знаходиться безпосередньо під коренем. Ці випадки включають подвійне обертання, яке реструктурує дерево, щоб привести x до кореня.
  + (zig-zag) випадок, коли x є правим нащадком свого батька, а батько є лівим нащадком прабатька , або навпаки. У цьому випадку функція виконує зигзагоподібне обертання, щоб перемістити x до кореня.

Після кожної ітерації циклу while батьківський елемент x буде перевірятися знову, доки x не стане коренем дерева.

* Метод findNode шукає в дереві задане значення Rational і повертає вузол, що містить це значення, або nullptr, якщо значення немає в дереві. Функція починається з кореня дерева та проходить вниз по дереву, порівнюючи значення вузлів із цільовим значенням. Якщо поточний вузол має те саме значення, що й цільове , воно повертається після його розширення. Якщо цільове значення менше значення поточного вузла, функція переміщується до лівого дочірнього елемента поточного вузла. Якщо цільове значення більше, ніж значення поточного вузла, функція переміщується до правого дочірнього елемента поточного вузла. Якщо поточний вузол має значення nullptr, функція повертає nullptr.
* Метод split розбиває даний вузол на два піддерева, одне з яких містить усі вузли, менші за задане значення, а інше містить усі вузли, які більше або дорівнюють заданому значенню. Функція спочатку перевіряє, чи вхідний вузол є nullptr. Якщо це так, він встановлює посилання ліворуч і праворуч, встановленими на покажчики лівого та правого піддерев на значення nullptr і повертає.

В іншому випадку метод split викликає функцію splay, щоб перевести вузол зі значенням val до кореня дерева.

Якщо значення кореневого вузла менше, ніж val, це означає, що всі вузли в лівому піддереві мають значення, менші за val. У цьому випадку функція встановлює ліворуч кореневий вузол, а праворуч правий дочірній вузол. Потім він від’єднує правий дочірній вузол від кореневого вузла, встановлюючи для його батьківського вузла значення nullptr, і для правого дочірнього вузла — nullptr.

З іншого боку, якщо значення кореневого вузла більше або дорівнює val, це означає, що всі вузли правого піддерева мають значення, більші або дорівнюють val. У цьому випадку функція встановлює праворуч кореневий вузол і ліворуч лівий дочірній вузол. Потім він від’єднує лівого дочірнього вузла від кореневого вузла, встановлюючи для його батьківського вузла значення nullptr, і для лівого дочірнього вузла — nullptr.

Функція повертається з двома посиланнями ліворуч і праворуч, встановленими на покажчики лівого та правого піддерев після розбиття. Нарешті, метод встановлює батьківські покажчики лівого та правого піддерев на nullptr.

* Метод злиття(merge) об’єднує два піддерева в одне дерево. Він приймає два вузли Splay-дерева як вхідні дані, лівий і правий, і повертає новий вузол Splay-дерева, що представляє об’єднане дерево.

Функція спочатку перевіряє, чи лівий або правий вузол є nullptr. Якщо будь-який з них дорівнює nullptr, функція просто повертає інший вузол.

Якщо жоден з вузлів не має значення nullptr, функція переходить вниз по крайній правій гілці лівого піддерева, щоб знайти вузол із максимальним значенням. Цей вузол стане новим коренем об’єднаного дерева. Функція викликає splay() на цьому вузлі, щоб перемістити його в корінь дерева.

Далі функція встановлює правий дочірній елемент максимального вузла як правий вузол. Нарешті, функція повертає максимальний вузол, який тепер є коренем об’єднаного дерева.

Клас SplayTree має публічні методи insert, remove, find, clear і print:

* Метод вставки (insert) вставляє задане значення Rational у дерево, зберігаючи баланс і порядок вузлів.Спочатку він перевіряє, чи вузол із таким самим значенням вже існує в дереві, викликаючи функцію findNode.

Якщо вузол із таким самим значенням існує, він оновлює значення вузла, а потім викликає функцію розширення на цьому вузлі. Це переносить вузол до кореня дерева для ефективного майбутнього доступу.

Якщо значення ще не існує в дереві, викликається функція розділення з коренем дерева та новим значенням .

Функція split розбиває дерево на два піддерева: одне з усіма вузлами, меншими за нове значення, і одне з усіма вузлами, більшими за нове значення.

Потім функція створює новий вузол із заданим значенням, і встановлює для його батьківського елемента значення nullptr, і для його лівих і правих дочірніх елементів — піддерева, створені на попередньому кроці.

Потім він оновлює батьківські покажчики лівого та правого дочірніх елементів нового вузла, щоб вони вказували на новий вузол.

Нарешті, функція оновлює корінь дерева як новий вузол.

* Метод видалення(remove) видаляє задане раціональне значення з дерева, зберігаючи баланс і порядок вузлів. Спочатку він знаходить вузол за допомогою функції findNode, а потім розгортає дерево так, щоб вузол, який потрібно видалити, став кореневим. Потім він розбиває дерево на дві частини: ліве піддерево кореня та праве піддерево кореня. Потім він об’єднує ці два піддерева разом, фактично видаляючи кореневий вузол з дерева. І видаляємо вузол із пам'яті.
* Метод пошуку( find) шукає в дереві задане значення Rational і повертає true, якщо значення є в дереві, і false в іншому випадку. Функція find спочатку викликає іншу функцію findNode з тим самим аргументом val для пошуку вузла, що містить указане значення в Splay-дереві.

Якщо findNode повертає ненульовий покажчик, find повертає true, вказуючи, що значення знайдено в дереві. В іншому випадку він повертає false, вказуючи, що значення не знайдено в дереві. Після того, як вершина знайдена, тягнемо її вгору і робимо коренем через процедуру splay.

* Метод очищення (clear) видаляє всі елементи з дерева, ітеративно видаляючи кореневий вузол за допомогою методу remove, доки дерево не стане порожнім.
* Метод друку (print) друкує значення всіх вузлів у дереві по порядку.

**Складність**

Часова складність наданої реалізації SplayTree залежить від операції, що виконується:

* Операції вставки та видалення мають часову складність O(log n), де n — кількість елементів у дереві. Це пояснюється тим, що ці операції вимагають пошуку правильної позиції в дереві для вставки або видалення елемента, а потім рзширювання дерева, щоб перевести новий вузол або наступника/попередника видаленого вузла в корінь дерева.
* Операція пошуку має таку ж часову складність, як і операції вставки та видалення, O(log n).
* Операція очищення має часову складність O(n), де n – кількість елементів у дереві. Це пояснюється тим, що для цього потрібно обійти все дерево та видалити кожен вузол.
* Операція друку має часову складність O(n), де n – кількість елементів у дереві. Це пов’язано з тим, що для цього потрібно обійти все дерево та роздрукувати кожен вузол.

# **Мова програмування**

С++

# 

# **Модулі програми**

**class** Rational //клас під назвою "Rational", який представляє раціональні числа.

**class** SplayTree //клас SplayTree, який представляє саморегулююче бінарне дерево пошуку, де операції пошуку, вставки та видалення елементів виконуються шляхом виконання операцій розширення(splay), які переміщують елемент, до якого здійснюється доступ, у корінь дерева. Відкриті функції класу SplayTree включають insert(), remove(), find(), clear() і print().Приватні функції класу SplayTree включають rotateLeft(), rotateRight(), splay(), findNode(), split() і merge(), які є допоміжними функціями, які використовуються публічними функціями для виконання необхідних операцій розширення , знаходити вузли в дереві, розділяти та об’єднувати піддерева.

**struct** Node //структура Node, представляє вузол у дереві, що містить значення Rational і покажчики на його батьківського, лівого та правого дочірніх елементів.

rotateLeft //функція rotateLeft для класу SplayTree. Метою цієї функції є обертання вузла та його правого дочірнього елемента вліво, роблячи правий дочірній елемент новим коренем піддерева.

rotateRight //функція rotateRight для класу SplayTree. Функція бере вказівник на об’єкт Node у Splay Tree і повертає його праворуч роблячи його лівий дочірній елемент новим коренем піддерева.

splay //Функція splay у класі SplayTree виконує операцію розширення на заданому вузлі x. Розширення — це операція, яка використовується в самобалансуючих бінарних деревах пошуку, де нещодавно доступний вузол переміщується до кореня дерева, щоб скоротити час доступу в майбутньому.

findNode //Ця функція шукає вузол у розширюваному дереві із заданим значенням . Він повертає вказівник на вузол, якщо він існує в дереві, і nullptr в іншому випадку. Він також застосовує операцію розширення до вузла, якщо його знайдено, або до останнього вузла, до якого зверталися, якщо не знайдено.

split //метод розбиття для класу SplayTree розбиває піддерево з коренем у вузлі на два піддерева: одне містить усі вузли зі значеннями, меншими за задане значення, а інше містить усі вузли зі значеннями, більшими або рівними задане значення.

merge //функція merge() для дерева SplayTree. Вона приймає два вузли дерева як вхідні дані, лівий і правий, і повертає новий вузол SplayTree, що представляє об’єднане дерево.

insert // функція insert() вставляє новий вузол зі заданим значенням у Splay Tree, з використанням функції split(розбиття).

remove // функція видаляє вузол із заданим значенням із розширюваного дерева.

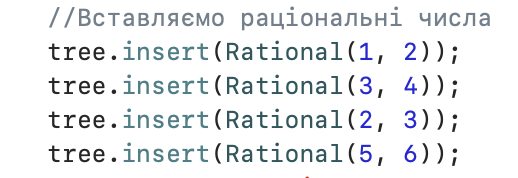
clear //Метод clear класу SplayTree видаляє всі елементи з дерева.

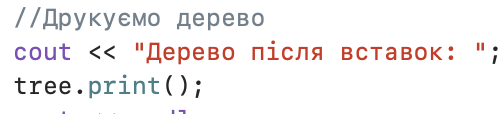
# **Інтерфейс користувача**

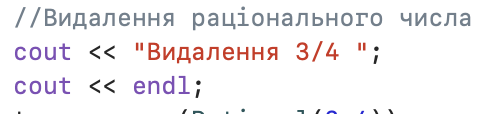
Вхідні дані задаються у програмі, а результат виводиться у консоль.

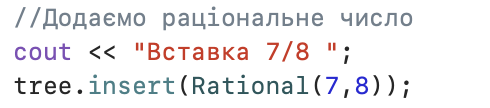
# 

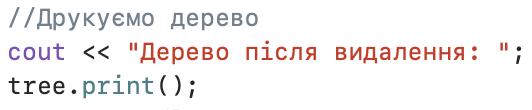
# **Приклад**

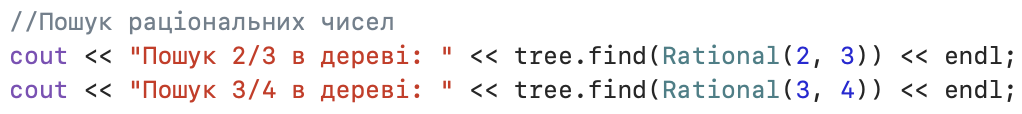


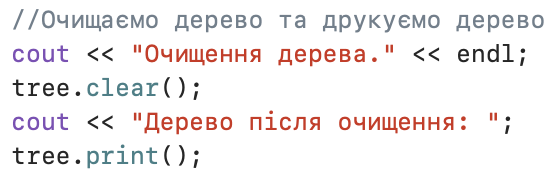




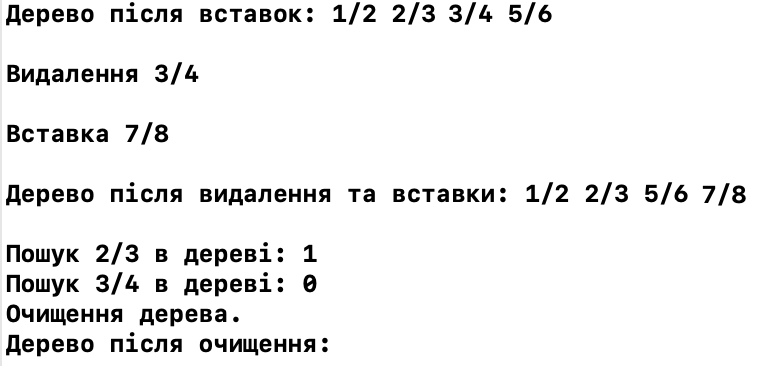








**Вивід програми :**



Програма демонструє реалізацію розширюваного дерева для раціональних чисел у C++. Програма спочатку вставляє 4 раціональних числа в дерево, а потім видаляє одне з них, 3/4. Після цього він вставляє нове раціональне число, 7/8, і друкує результуюче дерево. Далі програма виконує пошук 2/3 і 3/4 у дереві, друкуючи результати. Нарешті, програма очищає дерево, видаляючи всі його вузли, і друкує отримане порожнє дерево.

Вихідні дані програми показують, що реалізація розширюваного дерева працює правильно. Видалення 3/4 з дерева успішне, вставка 7/8 також успішна. Пошук 2/3 повертає 1, вказуючи, що число є в дереві, тоді як пошук 3/4 повертає 0, вказуючи, що число не знайдено у дереві. Програма також успішно очищає дерево, залишаючи його порожнім. Загалом програма демонструє корисність розширюваних дерев для ефективного пошуку та маніпулювання структурами даних, що містять раціональні числа.

**Висновок**

Підсумовуючи зазначимо, що реалізація розширюваного дерева для раціональних чисел у C++ є ефективним і практичним способом зберігання та маніпулювання раціональними числами. Splay tree — це самобалансуюче бінарне дерево пошуку, яке забезпечує швидкий доступ і ефективні операції з раціональними числами. Реалізація передбачає створення структури вузлів, яка містить чисельник і знаменник кожного раціонального числа, а також необхідні функції для вставки, видалення та пошуку вузлів у дереві.

Використовуючи реалізацію розширюваного дерева, ми можемо досягти часової складності O(log n) для більшості операцій, що робить його відповідною структурою даних для програм, які потребують ефективного та швидкого доступу до раціональних чисел.

Загалом, реалізація розширюваного дерева у C++ забезпечує потужний інструмент для ефективного зберігання, маніпулювання та доступу до раціональних чисел. Це надійна та ефективна структура даних, яку можна використовувати в широкому діапазоні програм, які потребують швидкого доступу до раціональних чисел.

# **Література**

1. https://wikipedia.org/wiki/Splay-дерево
2. https://habr.com/company/JetBrains-education/blog/210296/
3. Лекція з предмету «Алгоритми та складність »
4. https://www.scaler.com/topics/data-structures/splay-tree/