Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Завдання №2 (друга частина)

**“** **AA-дерево для комплексних чисел”**

Виконав студент 2-го курсу

Групи К-28

Адамов Олексій Віталійович

Київ 2022

**Завдання**

Реалізувати AA-дерево для комплексних чисел.

**Теорія**

AA-дерево - це варіант збалансованого дерева, а саме червоно-чорного дерева. До умов червоного-чорного дерева додається умова, що червоні вершини можуть бути лише правими потомками.

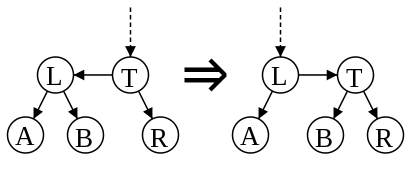
Але простіше перейти від кольорів до рівнів. З такими умовами:

1. Рівень кожного листа дорівнює одиниці.
2. Рівень кожного лівого сина рівно на один менше, ніж у його батька.
3. Рівень кожної правого сина дорівнює або на один менше рівня його батька.
4. Рівень кожного правого онука строго нижчий, ніж у його дідуся.
5. Кожен вузол, рівень якого більше одного, має два сини.

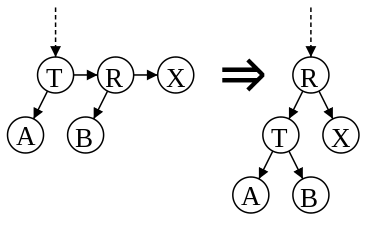
Горизонтальним називається зв’язок вузлів одного рівня.

Видалення, та додавання можуть порушити інваріанти AA-дерева. Для їх відновлення введено дві операції SKEW та SPLIT.

SKEW прибирає лівий горизонтальний зв’язок:



SPLIT виправляє ситувацію з двома правими горизонтальними зв’язками:



**Алгоритм**

SKEW та SPLIT реалізуються тривіально, причому так, щоб вони нічого не змінювали, якщо це не потрібно.

Додавання елемента:

1. Шукаємо (звичайним пошукому у бінарному дереві) місце для вставки та вставляємо туди елемент.
2. На шляху назад вгору для кожної вершини визиваємо спочатку SKEW, а потім SPLIT. Таким чином відновлюючи властивості AA-дерева.

Видалення елемента:

1. Знаходимо (звичайним пошукому у бінарному дереві) елемент який потрібно видалити.
2. Якщо це лист(немає лівого та правого сина) просто видаляємо його.
3. Якщо є лівий син, то знаходимо попередника (найправіший потомок лівого сина), копіюємо ключ попередника у вузол який хотіли видалити, та видаляємо попередника (рекурсивно).
4. Інакше є правий син. Знаходимо наступника (найлівіший потомок правого сина), копіюємо ключ наступника у вузол який хотіли видалити, та видаляємо наступника (рекурсивно).
5. На шляху назад вгору для кожної вершини T :
   1. Зменшуєм рівень T на одиницю, якщо він більше за найменьший рівень його синів + 1. Зменшуєм рівень правого сина T на одиницю, якщо він більше за рівень лівого сина T + 1.
   2. SKEW(T).
   3. SKEW(правого сина T).
   4. SKEW(правого сина правого сина T), якщо правий син T існує.
   5. SPLIT(правого сина T).
   6. SPLIT(T).

**Складність**

n - кількість елементів у множині.

Часова складність додавання, видалення, пошуку - O(log(n)).

Потрібна додаткова пам’ять на кожний вузол - його рівень (число).

**Мова програмування**

С++

**Модулі програми**

1.

class AA\_tree

{

private:

AA\_tree\_node\* root = nullptr;

public:

bool is\_in(Complex\_num key);

void insert(Complex\_num key);

void del(Complex\_num key);

void print();

};

AA-дерево з простим інтерфейсом додавання, видалення, пошуку, та виведення дерева у консоль.

2.

struct AA\_tree\_node

{

Complex\_num key;

unsigned int level;

AA\_tree\_node\* left;

AA\_tree\_node\* right;

// leaf with key

AA\_tree\_node(Complex\_num key);

bool is\_leaf();

static AA\_tree\_node\* del(AA\_tree\_node\* root, Complex\_num key);

static AA\_tree\_node\* insert(AA\_tree\_node\* root, Complex\_num key);

static bool is\_in(AA\_tree\_node\* root, Complex\_num key);

void print(std::vector<std::stringstream>& output, std::size\_t row);

private:

static AA\_tree\_node\* skew(AA\_tree\_node\* node);

static AA\_tree\_node\* split(AA\_tree\_node\* node);

static AA\_tree\_node\* decrease\_level(AA\_tree\_node\* node);

static AA\_tree\_node\* predecessor(AA\_tree\_node\* node);

static AA\_tree\_node\* successor(AA\_tree\_node\* node);

};

Класс для вузла AA-дерева. Саме тут основна реалізація алгоритму (Skew, SPIT, insert, delete).

3. struct Complex\_num

Структура для збереження, порівняння, виведення комплексних чисел.

**Інтерфейс користувача**

Вхідні дані вводяться з консолі і виводяться також у консоль.

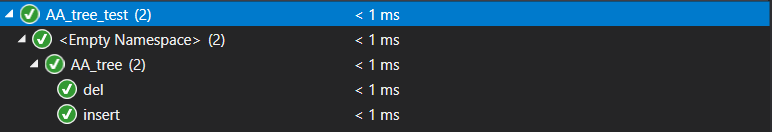
**Тестові приклади**

| **input** | **output** |
| --- | --- |
| ins (30, 2)  ins (70, 0)  ins (15, 4)  ins (85, 7)  ins (90, 1)  ins (80, 0)  ins (50, 50)  del (50, 50)  exit |  |

ins (90,1) - спочатку вставляється справа від (85,7), а потім за допомогою SPLIT це виправляється.

**Юніт-тести**

Програма проходить юніт-тести.



**Висновки**

AA-дерево збалансоване дерево, і виконує стандартні операції за логарифмічний час. По ефективності він майже еквівалентний до червоно чорного дерева. Хоча AA-дереву потрібно робити більше поворотів, але більш прості алгоритми працюють трохи швидше, що нівелює суттєву різницю у ефективності. Червоно чорне дерево працює більш страбільно, але AA-дерево частіше виходить більш гладким, що збільшує швидкість пошуку.

**Література**

* <https://en.wikipedia.org/wiki/AA_tree>
* Лекція №4 другого семестру з Алгоритмів та Складності (викладач - Шкільняк О.С.)**.**