Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Алгоритми та складність

Завдання №2

«Дерево порядкової статистики (на основі червоно-чорного дерева)»

Виконав студент 2-го курсу

Групи К-29

Пащенко Дмитро Вікторович

2020

**Завдання**:

Реалізувати дерево порядкової статистики на основі червоно-чорного дерева, застосувавши його до моделі, описаної у Вашому варіанті.

**Варіант 15:**

Предметна область: Залізниця

Об'єкти: Дороги, Станції

Примітка: Є безліч залізничних доріг. У відомстві кожної дороги знаходиться безліч станцій.

**Теорія:**

* Червоно-чорне дерево – бінарне дерево пошуку з додатковим бітом кольору в кожному вузлі: червоний чи чорний.
* Дерева наближено збалансовані: жоден шлях в червоно-чорному дереві не відрізняється від іншого більше ніж удвічі.
* У кожного дійсного вузла має бути два сина – дійсних чи nullptr. Останніх вважатимемо листками чорного кольору.
* Бінарне дерево пошуку буде червоно-чорним, якщо задовольнятиме червоно-чорні властивості.
* Червоно-чорні властивості
  + 1. Кожен вузол є або червоним, або чорним.
  + 2. Корінь дерева – чорний.
  + 3. Кожен лист дерева (nullptr) – чорний.
  + 4. Якщо вузол червоний, то обидва його сини чорні.
  + 5. Для кожного вузла всі шляхи від нього до листів-потомків містять однакову кількість чорних вузлів.
* i-та порядкова статистика множини з n елементів – елемент з i-м в порядку зростання ключем.
* Дерево порядкової статистики T (order-statistic tree) –червоно-чорне дерево з додатковим інформаційним полем size[x] (розмір піддерева з коренем x).
* Модифіковане червоно-чорне дерево дозволяє знайти порядкову статистику за час О(lg n).

**Мова програмування:** С++.

**Інтерфейс користувача**

Введення даних відбувається через консоль.

Вхідні дані: попередньо задане дерево порядкової статистики, що містить станції. Користувач вводить номер статистики – ціле невід’ємне число, що не перевищує розмір дерева.

Повертає: ім’я станції, що є шуканою статистикою, або повідомлення про помилку, якщо введені хибні дані.

**Модулі програми:**

* **void rotate(Node\* parent, Node\* child)**

Здійснює поворот сина навколо батька.

**Складність:** .

* **void insertFixup(Node\* current)**

Відновлює червоно-чорні властивості після вставки вершини

**Складність:** .

* **Node\* select(Node\* current, unsigned i) const**

Рекурсивно знаходить і-ту порядкову статистику.

**Складність:** , де n – кількість вершин дерева.

* **Node\* search(T const &key) const**

Шукає вершину з заданим ключем.

**Складність:** , де n – кількість вершин дерева.

* **void insert(T const &key)**

Вставляє вершину з заданим ключем.

**Складність:** , де n – кількість вершин дерева.

* **void decreaseUp(Node\* current)**

Зменшує кількість нащадків на одиницю у всіх вершин на шляху до кореня від заданої вершини (включно).

**Складність:** , де n – кількість вершин дерева.

* **void removeFixup(Node\* current)**

Відновлює червоно-чорні властивості після видалення вершини.

**Складність:** .

* **void remove(T const &key)**

Видаляє вершину з заданим ключем.

**Складність:** , де n – кількість вершин дерева.

* **T operator[](unsigned i) const**

Повертає і-ту порядкову статистику.

**Складність:** , де n – кількість вершин дерева.

* **void print(Node\* current, unsigned level = 0, Status stat = Status::ROOT) const**

Рекурсивно порівнево друкує дерево, починаючи з заданої вершини.

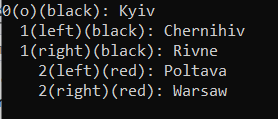
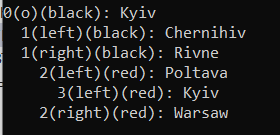
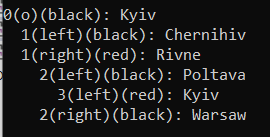
**Складність:** , де n – кількість вершин дерева.

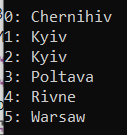
* **void print() const**

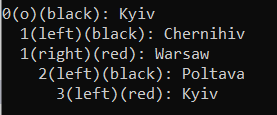
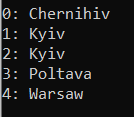
Друкує дерево, обгортка для попередньої функції.

**Складність:** , де n – кількість вершин дерева.

**Тестові приклади**

1. Створив порожнє дерево. Поступово додав 5 станцій. Бачимо, що властивості бінарного дерева пошуку та червоно-чорні властивості виконуються.
2. Додав нову вершину Kyiv. Спочатку вона встановилася як лівий син вершини Poltava, тобто як у звичайне бінарне дерево пошуку, причому колір встановився червоний.
3. Маємо перший випадок конфігурації: дідусь (Rivne) чорний, дядько (Warsaw) червоний. Відбувається перефарбування вершин: дід стає чорним, батько і дядько чорними. Переходимо до дідуся.
4. Бачимо, що дідуся в поточної вершини Rivne немає, завершуємо алгоритм.
5. Виводжу на екран усі порядкові статистики по-черзі. Бачимо, що порядок вірний.



1. Видаляю вершину Rivne. Алгоритм шукає найменший елемент більший за Rivne. Це Warsaw. У рівне копіюється ключ Warsaw, вершина Warsaw видаляється.
2. У Warsaw не було правого нащадка, тому алгоритм завершує роботу – виправляти нічого.
3. Знову виводжу всі порядкові статистики на екран. Переконуємося, що порядок правильний.

**Література**

* Кормен, Лейзерсон, Рівест, Штайн. Алгоритми: побудова і аналіз, 2-е видання. – 2005
* Лекція №2, Шкільняк О. С.