Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Завдання №5

“Реалізація персистентної множини на основі

червоно-чорного дерева”

Варіант 15

Виконав студент 2-го курсу

Групи К-29

Пащенко Дмитро Вікторович

2020

**Завдання**:

Реалізувати персистентну множину на основі червоно-чорного дерева, застосувавши її до моделі, описаної у Вашому варіанті. Час роботи вставки і видалення в найгіршому випадку і  об‘єм необхідної пам’яті  мають бути О(log  n).

**Варіант 15:**

Предметна область: Залізниця

Об'єкти: Дороги, Станції

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

**Теорія:**

* **Персистентні динамічні множини:**
  + Зберігають свої попередні версії (і доступ до них) в процесі внесення змін.
  + Може зберігатися тільки остання версія або всі існуючі попередні.
  + Персистентними можна зробити різні структури даних.
  + Для ефективної реалізації просте копіювання не підходить.
  + Розглянемо реалізацію персистентної множини з операціями пошуку, видалення та вставки на основі червоно-чорного дерева.
  + Для кожної версії множини зберігається свій корінь.
  + Фактично будується копія лише тієї вітки (шляху), де можуть відбутися зміни.
* **Часткова персистентність**: до кожної версії можна робити запити, але змінювати можна тільки останню версію структури даних.
* **Повна персистентність**: можна змінювати не тільки останню, але і будь-яку версію структур даних, також до будь-якої версії можна робити запити.

**Алгоритм**

Складність реалізації для червоно-чорного дерева полягає в тому, що недостатньо враховувати зміни лише ланцюга, що проходить від кореневої вершини до вершини, яка вставлена або буде видалена. У процесі відновлення червоно-чорних властивостей можуть бути змінені ще й суміжні з ланцюгом вершини й не тільки вони. Алгоритм базується на глибокій копії суміжних з ланцюгом вершин. Для операції вставки це копія глибини 2 (тобто копіюється вершина + дочірні їй), для видалення - 3. Цих глибин виявляється достатньо для покриття можливих поворотів та змін кольору.

**Складність алгоритму**

Операції вставки та видалення мають логарифмічну складність. Копіювання ланцюга також має логарифмічну складність, оскільки на кожну його вершину витрачається часу на копіювання її та глибоке копіювання суміжної вершини. Нова пам’ять виділяється лише під ланцюг та глибокі копії суміжних йому вершин – тому витрати пам’яті також логарифмічні.

**Складність**: .

**Витрати пам’яті**: .

**Мова програмування:** С++.

**Інтерфейс користувача**

Введення даних відбувається через консоль.

Вхідні дані: команди

* 0 – завершити програму
* 1 – додати залізницю (за назвою)
* 2 – видалити залізницю (за назвою)
* 3 – вивести дерево (preorder)
* 4 – вивести стан дерева n-кроків тому (n – беззнакове ціле)

**Модулі програми:**

* **Node\* PersistentRBTree<T>::copy(Node const &source)**

Копіює вершину, змінює покажчики в дітей.

**Складність:** .

* **Node\* PersistentRBTree<T>::deepCopy(Node const &source, unsigned depth)**

Копіює вершину до вказаної глибини.

**Складність:** .

* **Node\* PersistentRBTree<T>::searchCopying(T const &key)**

Шукає заданий ключ, копіюючи всі вершини на шляху.

**Складність:** .

* **void PersistentRBTree<T>::insert(T const &key)**

Модифікація звичайної вставки (копіює ланцюг від кореня до нової вершини, а також приєднує до нього копії суміжних вершин).

**Складність:** .

* **void PersistentRBTree<T>::remove(T const &key)**

Модифікація звичайного видалення (копіює ланцюг від кореня до вершини, що слідує за видаленою, а також приєднує до нього глибокі копії суміжних вершин).

**Складність:** .

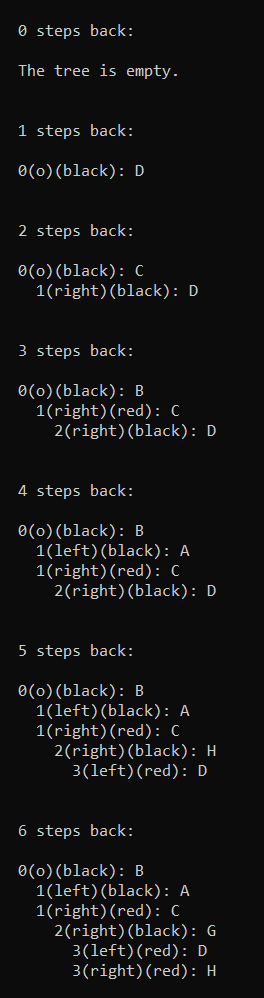
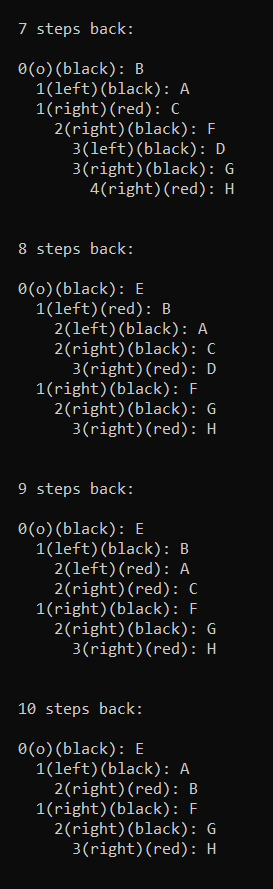
* **void PersistentRBTree<T>::printPrevious(size\_t steps\_back)**

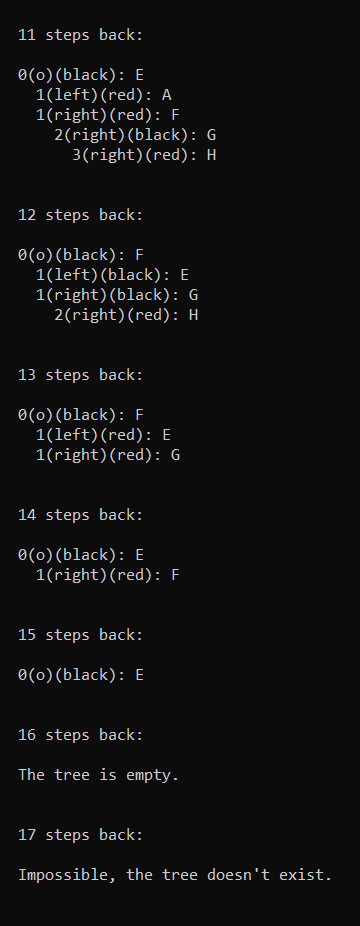
Друкує версію дерева вказану кількість кроків назад.

**Складність:** .

**Тестові приклади**

1. Для зручності всі залізниці називав великими латинськими буквами. Спочатку додав 4 залізниці від «E» до «H», потім від «А» до «D». Потім видалив у такому ж порядку. Вивів усі стани дерева, починаючи від кінцевого. Бачимо, що ніяких втрат інформації немає. Читаючи в оберненому порядку, прослідковуємо вірний хід вставок та видалень.





**Висновки**

Успішно реалізував персистентну множину на основі червоно-чорного дерева. Вдалося досягти повної (але без можливості модифікації попередніх версій) персистентності, не змінивши складностей вставки та видалення та використавши логарифмічну кількість пам’яті.

**Література**

* <https://habr.com/ru/post/142572/>