Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Завдання №5

“ Персистентна множина на основі червоно-чорного дерева”

Виконав студент 2-го курсу

Групи К-29

Демиденко Андрій Геннадійович

2021

**Предметна область**

Предметна область: Відділ кадрів

Об’єкти: Підрозділи, Співробітники

Примітка: Існує множина підрозділів. Кожен підрозділ включає в себе множину співробітників.

**Завдання**

Реалізувати персистентну множину на основі червоно-чорного дерева. У вузлах дерева будемо зберігати інформацію про підрозділи.

**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 бінарного червоно-чорного дерева пошуку. Для кожної версії множини зберігається свій корінь. Фактично будується копія лише тієї вітки (шляху), де відбулися змін.

**Алгоритм**

Insert:  
Стійкий характер дерева проявляється в реалізації insert. Замість того, щоб змінювати існуюче дерево, insert створюється нове дерево з новим елементом, вставленим у потрібне місце. Реалізація є рекурсивною, тому уявіть, що ви знаходитесь на піддереві великого дерева. Це піддерево може бути порожнім. Вставка елемента в порожнє дерево означає створення дерева з одним вузлом з вбудованим значенням x і двома пустими дітьми. Додавання вузла у не пусте дерево перестворює тільки вершини по шляху до змінюваному елементу (якщо іншого не вимагає протокол балансування). Далі іде відновлення властивостей червоно-чорного дерева за правилами.

Наприклад вставка ключа E у персистентне дерево

Початкове дерево

**A**

**C**

**F**

**D**

**B**

**G**

**H**

Після вставки

**A**

**C**

**F**

**D**

**B**

**G**

**C**

**F**

**D**

**G**

**H**

**E**

Усі червоно-чорні властивості не порушилися

Delete

Замість того, щоб змінювати існуюче дерево, Delete створює нове дерево з уже відсутнім елементом, а на його місце буде вставлено правий дочірній вузол, або якщо він пустий (nullptr ), то лівий. Далі іде відновлення властивостей червоно-чорного дерева. Усі ці повороти виконуються тільки на вузлах, які були скопійовані до нової версії дерева, а не просто взяті туди.

Видалимо елемент F з персистентної множини

**A**

**C**

**F**

**D**

**B**

**G**

**C**

**F**

**D**

**G**

**H**

**E**

Після видалення

**A**

**C**

**F**

**D**

**B**

**G**

**C**

**F**

**D**

**G**

**H**

**E**

**E**

**D**

**G**

**H**

Після відновлення властивостей

**A**

**C**

**F**

**D**

**B**

**G**

**C**

**F**

**D**

**G**

**H**

**E**

**E**

**D**

**G**

**H**

**Складність**

Виконання цих інваріантів гарантує, що дерево буде збалансованим за висотою log n, де n - кількість вузлів в дереві. Іншими словами, основні операції над червоно-чорного дерева гарантовано виконуються за логарифмічний час. Тобто складність .

**Мова програмування**

С++

**Модулі програми**

/\*

Class implements persistent set based on red-black tree

time of insertion, deletion and needed memory - O(log n) in worst case

\*/

template <typename T>

class PersistentSet

{

private:

//Some addition functions and classes

std::vector<Node\*> roots;

public:

//Constructor

PersistentSet();

//Insertion

void Insert(T& t);

//Removing

void Remove(const T& t);

//Output in console

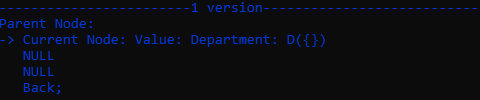
void Print();

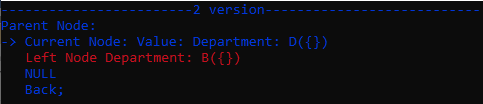
};

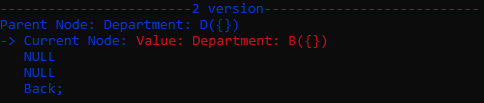
**Інтерфейс користувача**

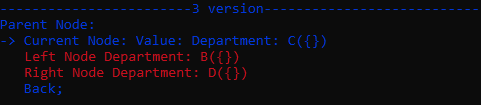
Вхідні дані генеруються програмою і виводяться у консоль.

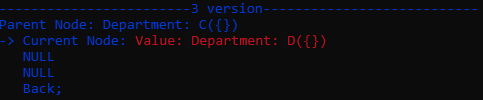
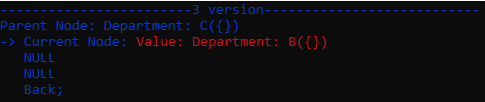
**Приклад виводу програми**

****

****

****

****

****

**Висновок**

При операціях над деревом, операція копіювання дійсно присутня, але в самих мінімальних масштабах. Більш формально - копіюється лише те, що дійсно змінюється. Ті частини, які не зазнали змін залишаються на місці і поширюються між версіями. Ці дві пропозиції - ключі до розуміння чисто функціональних структур даних. Зайва пам'ять, яка використовується для підтримки персистентності, компенсується можливістю отримати доступ до будь якої попередньої версії дерева.

**Література:**

* Лекція № 4
* [**https://habr.com/ru/post/208918/**](https://habr.com/ru/post/208918/)