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-28
Гуща Дмитро Сергійович

2021

Предметна область

Варіант 4
Предметна область: Учбовий відділ
Об’єкти: Групи, Студенти
Примітка: Маємо множину учбових груп. Кожна група містить в собі множину студентів

Завдання

Реалізувати персистентну множину на основі червоно-чорного дерева

Теорія

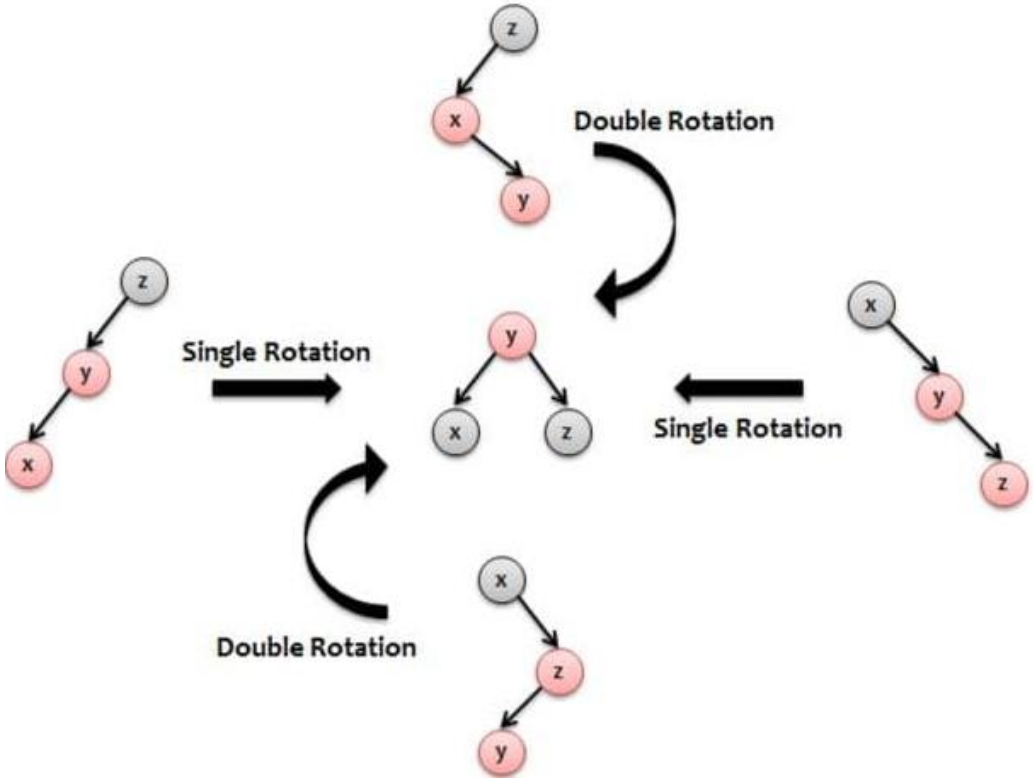
Персистентні динамічні множини

- 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. Перестворює тільки вершини по шляху до змінюваному елементу (якщо іншого не вимагає протокол балансування) Далі іде відновлення властивостей червоно-чорного дерева за правилами



Усі ці повороти виконуються тільки на вузлах які були скопійовані до новоїверсії дерева а не просто взяті туди

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

Складність

Виконання цих інваріантів гарантує, що дерево буде збалансованим з висотою log n, де n - кількість вузлів в дереві. Іншими словами, основні операції над ЧЧД гарантовано виконуються за логарифмічна час. Тобто складність $O(\log n)$

Мова програмування

C++

Модулі програми

student.h

```
class Student{}; //Класс опису студента
std::string getName(); // метод повертає ім’я студента
void getStudent();//метод виводить ID та ім'я студента в консоль
void setName(std::string name); //метод змінює ім'я студента
```

group.h

```
class Group {};
Group() : title("NULL"); //конструктор пустої групи
```

```
Group(std::string title); //конструктор з початковою назвою групи
Group(std::string title, Student* first_student); //конструктор з початковою назвою
групи та першим студентом
std::string getGroupTitle(); //модуль повертає назву групи
std::vector<Student*> getGroupStudents(); //модуль повертає множину студентів
void setGroupTitle(std::string title); //модуль змінює назву групи
void setGroupStudents(std::vector<Student*> students); //модуль змінює множину
студентів
void addStudent(Student* student); //додати нового студента
void printStudents(); //вивід у консоль усіх студентів групи
```

persistentTree.h

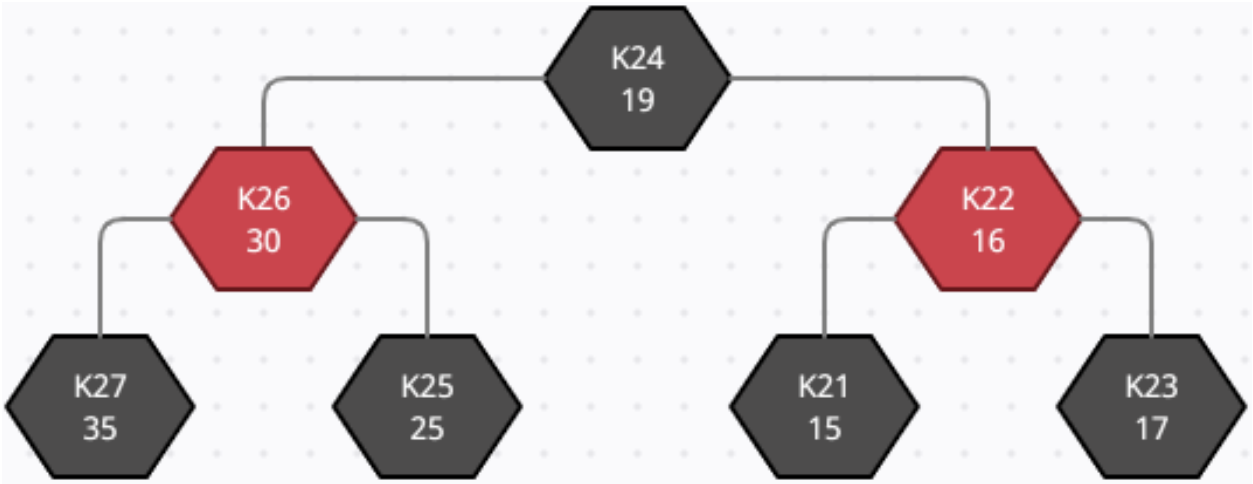
```
class PersistentSet//клас для опису персистентної множини
class Node//класс для опису вузла червоно-чорного дерева
Node()//конструктор червно-чорного дерева
void leftRotate(Node* toRotate)//функція робить лівий поворот для дерева
void rightRotate(Node* toRotate)//функція робить правий  поворот для дерева
void deleteNode(Node* toDelete)// функція для видалленя вузла дерева
void deleteFix(Node* toFix, bool& sideRight)//відновлення властивостей після
видалення
void backupTree(Node* toBackup, Node* parent)//перехід до попередній версії
червоно-чорного дерева
void print(Node* node, int level, bool left) const//функція для виводу червоно-
чорного дерева
void insert(T& t)//функція вставки у персистентне червоно-чорне дерево
void print()//функція виводу усіх версій дерева на консоль
```

Інтерфейс користувача

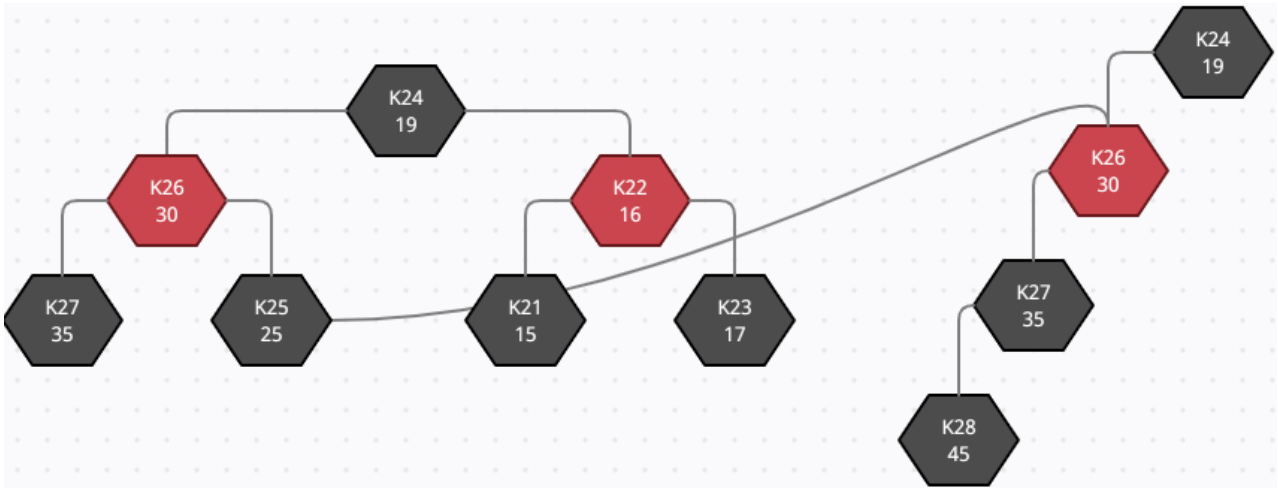
Вхідні дані генеруються програмою а вихідні дані виводяться у консоль.

Тестовий приклад

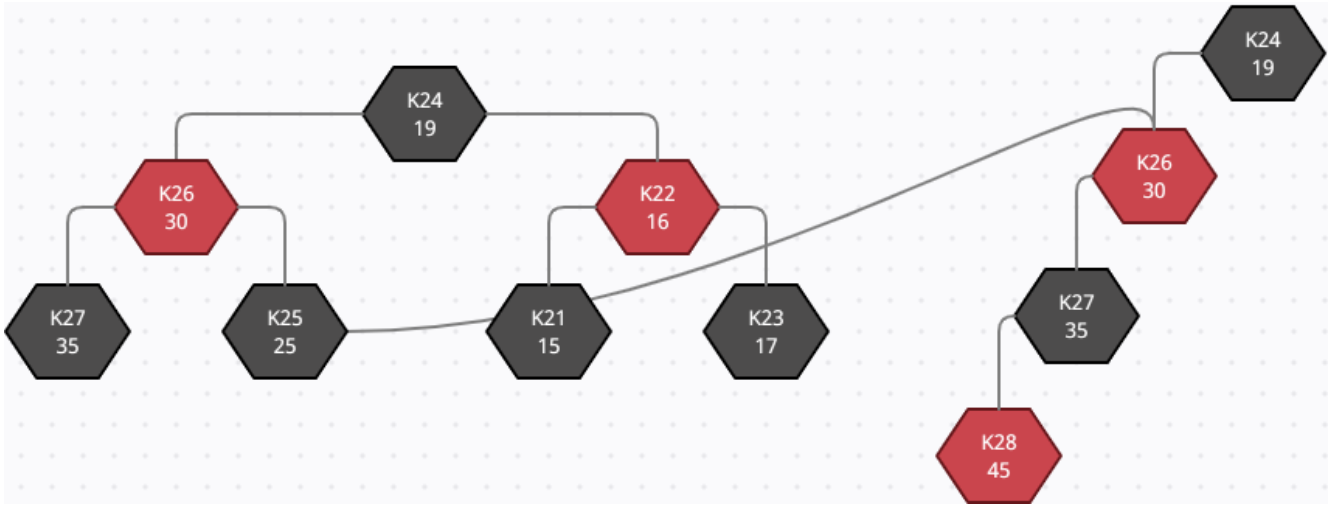
Початкове дерево



Після вставки вузла K28 45



Після перефарбовки



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 зміни 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/**