**Київський національний університет імені Тараса Шевченка**

**Факультет комп’ютерних наук та кібернетики**

**Алгоритми та складність**

**Завдання № 3**

**Звіт**

**Виконав:**

студент групи К-29

Грищенко Юрій Анатолійович

**Київ-2020**

**Умова задачі**

Реалізуйте розширюване дерево (splay tree).

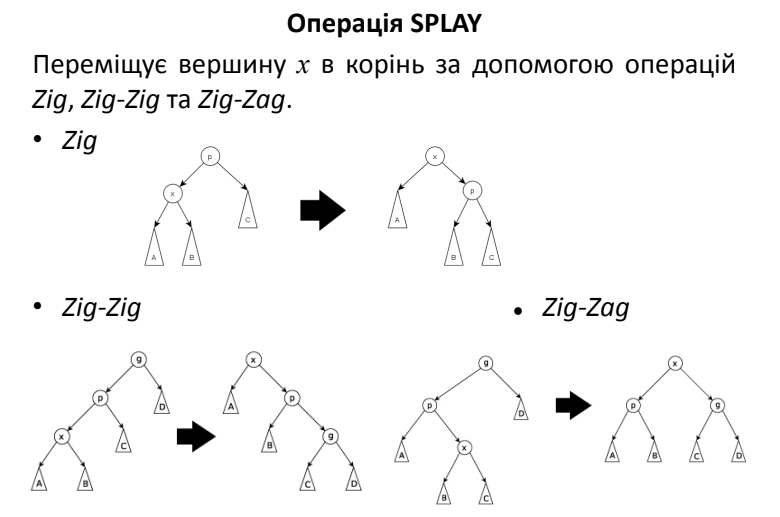
Предметна область: відділ кадрів (варіант 3). Обєкти: відділення фірми, працівники. Маємо множину відділень, у кожному відділенні зберігається множина працівників.

**Опис алгоритмів**

Припустимо, що нам необхідно мати відсортований список працівників фірми, в алфавітному порядку. Додавання, видалення, та пошук елементів для такого списку повинно бути швидким. У такому випадку можна використати **splay tree (розширюване дерево)**.

**Splay tree -** двійкове дерево пошуку з підтримкою збалансованості.

* Не потребує додаткових полів у вузлі, отже ефективно використовуює пам’ять.
* Явні функції балансування відсутні.
* При кожному звертанні до дерева виконується **«операція розширення»** (splay operation). В результаті вузли, до яких звертаються частіше, зберігаються ближче до кореня, а до яких рідше – ближче до листків.
* Завдяки цьому всі операції потребують часу в середньому О(lg n).



**Основні операції:**

* **Merge (об'єднання двох дерев)**. Для злиття дерев T1 і T2, в яких всі ключі T1 менше ключів в T2, робимо Splay для максимального елементу T1, тоді біля кореня T1 не буде правого дочірнього елемента. Після цього робимо T2 правим дочірнім елементом T1.
* **Split (розділення дерева на дві частини)**. Для розділення дерева знаходиться найменший елемент, більший або рівний x і для нього робиться Splay. Після цього відрізаємо ліве піддерево у якості другого дерева.
* **Search (пошук елемента)**. Спочатку звичайний пошук. При знаходженні елементу запускаємо Splay для нього.
* **Insert (додавання елемента)**. Запускаємо Split від елементу, що додається, і підвішуємо дерева, що вийшли, за нього.
* **Delete (видалення елемента)**. Знаходимо елемент в дереві, робимо Splay для нього, робимо поточним деревом Merge його дітей.

**Модулі програми:**

У своїй програмі я розбив модулі на дві категорії: ті, що стосуються саме предметної області (HumanResources.\*) і ті, що стосуються дерева порядкової статистики (SplayTree.hpp)

SplayTree<T> реалізовано як template-клас, тобто він може зберігати обєкти будь-якого класу. Єдина умова: для класу Т мають бути реалізовані оператори порівняння (>. ≥, <. ≤), за допомогою яких будуть сортуватися елементи.

Основні операції над деревом:

* **Додавання елемента** (void SplayTree<T>::insert(T value))
  + value стає коренем дерева.
* **Пошук елемента** (bool SplayTree<T>::contains(const T& value))
  + Якщо функція знаходить потрібний вузол, повертає true, інакше — false. Знайдений елемент стає коренем дерева.
* **Видалення елемента** (bool SplayTree<T>::remove(const T& value))
  + Якщо функція знаходить потрібний вузол, видаляє його і повертає true, інакше — false.
* **Розділення дерева** (SplayTree<T> SplayTree<T>::split(const T& value))
  + Розділення відносно value. Повертає нове дерево, в якому всі елементи менші за value. У поточному дереві залишаються тільки елементи більші або рівні за value.
* **Об’єднання дерев** (void SplayTree<T>::mergeGreater(SplayTree<T>& tree))
  + Додає до поточного дерева всі елементи tree. Кожен із них повинен бути більшим за всі елементи поточного дерева. tree стає пустим.

**Інтерфейс користувача, тестові приклади:**

У програмі є інтерактивний режим, який дозволяє користувачеві додавати, видаляти, шукати елементи, а також бачити структуру дерева. Після операцій додавання та видалення всі елементи дерева виводяться в алфавітному порядку. Також є допоміжна команда help.

Available commands: add, remove, find-by-name, help, print, exit

> **add**

Enter employee name: **Yurii**

{ { Name: Yurii } }

> **add**

Enter employee name: **Bob**

{ { Name: Bob }, { Name: Yurii } }

> **add**

Enter employee name: **Carl**

{ { Name: Bob }, { Name: Carl }, { Name: Yurii } }

> **print**

{ Name: Carl }

Left:

{ Name: Bob }

Right:

{ Name: Yurii }

> **add**

Enter employee name: **Alice**

{ { Name: Alice }, { Name: Bob }, { Name: Carl }, { Name: Yurii } }

> **print**

{ Name: Alice }

Right:

{ Name: Bob }

Right:

{ Name: Carl }

Right:

{ Name: Yurii }

> **find-by-name**

Enter employee name: **Carl**

Tree contains Carl? true

> **print**

{ Name: Carl }

Left:

{ Name: Bob }

Left:

{ Name: Alice }

Right:

{ Name: Yurii }

> **remove**

Enter employee name: **Bob**

{ { Name: Alice }, { Name: Carl }, { Name: Yurii } }

> print

{ Name: Alice }

Right:

{ Name: Carl }

Right:

{ Name: Yurii }

> **find-by-name**

Enter employee name: **Yurii**

Tree contains Yurii? true

> **print**

{ Name: Yurii }

Left:

{ Name: Carl }

Left:

{ Name: Alice }

> **exit**

**Список використаних джерел:**

* Лекція 3 з курсу “Алгоритми та складність” Шкільняк О.С.