Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Лабораторна робота №4

Виконав: Олексишин Олексій

Група К-28

Кафедра інтелектуальних програмних систем

**2020**

**Предметна область**

Вариант 3

Предметная область Отдел кадров Объекты Подразделения, Сотрудники Примечание Имеется множество подразделений предприятия. В каждом подразделении работает множество сотрудников.

Для того щоб дану структуру даних можна було помістити в дерево я перевантажив operator< (та інші оператори порівняння), для виведення в консоль operator<<.

**Завдання**

Оптимальне бінарне дерево пошуку.

**Алгоритм і теорія**

Нехай в нас є множина S = {*e*1, *e*2, …, *en*}, для якої *e*1 < *e*2 < … < *en.* Розглянемо бінарне дерево пошуку, яке складається з елементів S. Чим частіше треба робити запит якогось елемента, тим ближче він має бути до кореня. Варітсю *cost* доступа до елемента *ei* з множини S в дереві будемо називати значення *cost*(*ei*), рівне числу ребер на шляху від кореня до даного елемента. Нехай ми маємо частоту з якою ми робимо запити до кожного елемента з S – (f(*e*1), f(*e*2), ..., f(*en*)), визначимо загальну варість дерева наступним чином:

*f*(*e*1) \* *cost*(*e*1) + *f*(*e*2) \* *cost*(*e*2) + … + *f*(*en*) \* *cost*(*en*)

Дерево, яке має найменшу вартість, вважається найкращим бінарним деревом пошуку для даної множини елементів. Тому воно називається Оптимальним Бінарним Деревом пошуку.

Нехай дорівнює вартості оптимального дерева пошуку, котре можна побудувати із елементів , , ..., . Очевидно, що . Для має місці рекурентність:

Елемент ставимо в корені. Вартість побудови лівого піддерева рівна , правого При цьому оскільки лівого піддерева знаходиться на один рівень нижче , то для врахування вартості лівого піддерева необхідно додати суму частот всіх його елементів, тобто значення

Аналогічно при підрахунку вартості правого піддерева необхідно врахувати

При покладемо .

Варто відмітити, що розв’язання задачі про оптимальне дерево пошуку аналогічне розв’язку задачі про оптимальне множення матриць.

**Складність алгоритму**

Для знаходження оптимального дерева нам треба знайти всі для *0 ≤ i,j ≤ n*. Тому складність алгоритму буде *O(n²).*

**Реалізація (мова С++)**

Програма реалізує клас PersistentSet, що містить в собі адаптовану реалізацію червоно-чорного дерева, методи для роботи з множиною та звичайний інтерфейс для роботи з множиною (insert, remove, search) та процедури для перебудови дерева.

**Основні модулі програми**

У файлі PersistentSet.hpp показаний інтерфейс вказаних класів, та реалізовані методи класів та процедури для роботи множиною та червоно-чорним деревом. Файл struct.hpp містить реалізацію класів предметної області. В файлі main.cpp міститься інтерфейс користувача та приклад роботи з реалізованою структурою даних.

Код можна знайти в моєму репозиторії на github за посиланням <https://github.com/ooleksyshyn/semester4/tree/master/algorithms/lab5>

**Інтерфейс користувача та приклади**

Програма дозволяє взаємодіяти з класом через простий інтерфейс – вставка, видалення, пошук елемента за ключем та за індексом. В коді є наведений приклад використання програми.

**Літературні джерела**

* [https://site.ada.edu.az/~medv/acm/Docs%20e-olimp/Volume%2016/1522.htm](https://site.ada.edu.az/~medv/acm/Docs e-olimp/Volume 16/1522.htm)