Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Звіт

лабораторної роботи №4

«Оптимальне дерево бінарного пошуку»

з предмету «Алгоритми та складність»

Над роботою працював:

студент 2 курсу

групи К-29

Маханько Ростислав

2019

**Завдання**. Реалізувати оптимальне бінарне дерево пошуку, дерево де для кожної вершини відомо частоту запитів для неї і відповідно до цього нам треба побудувати дерево таким чином, щоб загальна «вартість» операцій була мінімальною.(Див. задачу <https://www.e-olymp.com/uk/problems/1522>)

**Вхідні дані**: У першому рядку файлу з вхідними даними міститься число команд. Далі для кожної команди її назва, кількість гравців та їх імена, кожне значення з нового рядка. З клавіатури зчитуємо частоту запитів для кожної команди.

**Вихідні дані:** На екран виводиться представлення оптимального дерева бінарного пошуку.

**Алгоритм розв’язання**: Оптимальне дерево бінарного пошуку (Optimal BST), яке іноді називають двійковим деревом з балансом ваги, є двійковим деревом пошуку, яке забезпечує найменший можливий час пошуку (або очікуваний час пошуку) для заданої послідовності доступу.

Дано відсортовані масиви keys[0 .. n-1] пошукових ключів і frequencies[0 .. n-1] частот, де frequencies [i] - кількість запитів до ключа i, n – кількість ключів. Необхідно побудувати двійкове дерево пошуку всіх ключів таким чином, щоб загальна вартість всіх пошуків була якомога меншою.

Давайте спочатку визначимо вартість BST. Вартість вузла - це рівень цього вузла, помножений на його частоту. Рівень кореня = 1.

Оптимальна вартість dp для частот [l..r] може бути рекурсивно обчислена за допомогою наступної формули:

Потрібно обчислити dp[0][n-1], щоб отримати відповідь для всіх ключів.

Ми один за одним візьмемо всі вузли як корінь. Коли ми робимо k-й вузол кореневим, ми рекурсивно обчислимо оптимальну вартість від l до k - 1 і k + 1 до r. Додаємо суму частот від l до r, тому що кожен пошук буде проходити через кореневу систему і буде проводитися одне порівняння для кожного пошуку.

Якщо ми будемо додатково запам’ятовувати dp[l][r], то складність даного алгоритму буде складати O(n^3). Ми викликаємо функцію обчислення dp[l][r] не більше одного разу, тобто це O(n^2) викликів, і при кожному з них перебираємо k від l до r, що дає ще O(n) операцій.

**Використана література**:

<https://www.geeksforgeeks.org/optimal-binary-search-tree-dp-24/>

<https://en.wikipedia.org/wiki/Optimal_binary_search_tree>