Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Алгоритми та складність

Завдання №4

“Оптимальне бінарне дерево пошуку (динамічне програмування)”

Виконав студент 2-го курсу

Групи К-29

Пащенко Дмитро Вікторович

2020

**Завдання**:

Реалізувати оптимальне бінарне дерево пошуку засобами динамічного програмування, застосувавши його до моделі, описаної у Вашому варіанті.

**Варіант 15:**

Предметна область: Залізниця

Об'єкти: Дороги, Станції

Примітка: Є безліч залізничних доріг. У відомстві кожної дороги знаходиться безліч станцій.

**Теорія:**

* Розглянемо довільну множину S = {k1, k2, ..., kn}, що складається з n різних елементів таких, що k1 < k2 < ... < kn. Розглянемо бінарне дерево пошуку, що складається з елементів S. Чим частіше проводиться запит до елементу, тим ближче він повинен розташовуватися до кореня.
* Для кожного ключа задана pi – вірогідність його пошуку, i = . Причому шукані користувачем елементи не обов’язково містяться в дереві, тому передбачені ймовірності qi пошуків проміжних значень, i = , та самі значення di, i = .
* Припустимо, що фактична вартість пошуку визначається кількістю перевірених вузлів, тобто глибиною вузла, збільшеною на одиницю. Тоді мат. сподівання вартості пошуку в дереві з елементами з S рівне:

.

* Дерево, яке має найменшу вартість, вважається найкращим для пошуку елементів з S. Саме тому воно називається оптимальним бінарним деревом пошуку.
* Динамічне програмування є водночас і методом математичної оптимізації і методом комп'ютерного програмування. В обох контекстах воно використовує підхід спрощення пошуку розв'язку складної задачі, розбиттям її на простіші підзадачі, часто методом рекурсії.
* Хоча деякі задачі не можуть бути розв'язані таким чином, рішення, які охоплюють кілька точок у часі дійсно часто розбиваються рекурсивно на підзадачі. Подібно до цього, в комп'ютерних науках про проблему, яка може бути розбита на підзадачі рекурсивно, говорять що вона має оптимальну підструктуру.
* Оптимальна підструктура для даної задачі: якщо до складу оптимального дерева пошуку Т входить піддерево Т’, що містить ключі ki, …, kj, то це дерево має бути оптимальним для підзадачі з ключами ki, …, kj та фіктивними ключами di-1, …, dj.

**Мова програмування:** С++.

**Інтерфейс користувача**

Введення даних відбувається через консоль.

Вхідні дані: відсортований масив назв залізниць, імовірностей пошуку цих значень та імовірностей пошуку проміжних значень.

Результат: оптимальне дерево назв залізниць (друковане в preorder).

**Модулі програми:**

* **std::vector<std::vector<unsigned>> optimize(std::vector<double> const &true\_probs, std::vector<double> const &false\_probs)**

Вираховує оптимальну конфігурацію дерева, тобто таку, що мат. сподівання глибини пошуку мінімальне. Повертає масив коренів оптимальних піддерев.

**Складність:** .

* **void buildSubtree(std::vector<T> const &keys, std::vector<std::vector<unsigned>> const &roots, unsigned left, unsigned right)**

Рекурсивно будує оптимальне піддерево за масивом коренів.

* **void buildTree(std::vector<T> const &keys, std::vector<double> const &true\_probs, std::vector<double> const &false\_probs)**

Будує оптимальне дерево.

**Складність:**

* **void insert(T const &key)**

Вставляє в дерево вершину з даним ключем.

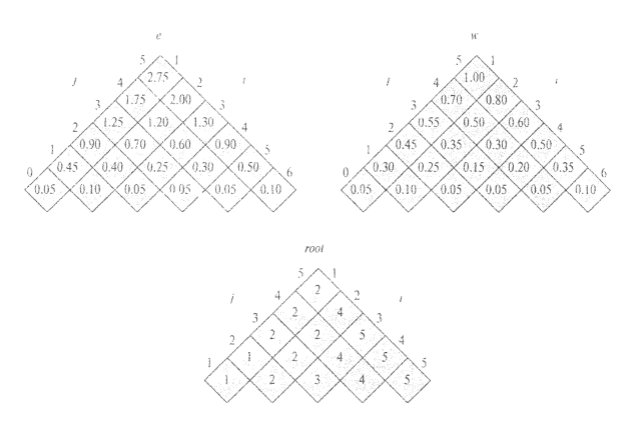
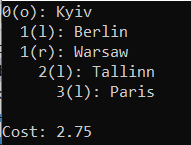
**Складність:**

* **void print(Node<T>\* current, unsigned level = 0, Status stat = Status::ROOT) const**

Друкує дерево (preorder).

**Складність:**

**Тестові приклади**

1. Створив масив залізниць S = < “Berlin”, “Kyiv”, “Paris”, “Tallinn”, “Warsaw” >. Створив масив ймовірностей пошуку кожного значення P = < 0.15, 0.10, 0.05, 0.10, 0.20 >. Створив масив ймовірностей пошуку проміжних значень Q = < 0.05, 0.10, 0.05, 0.05, 0.05, 0.10 >. Передав дані в конструктор дерева.
2. Алгоритм обчислив в точності такі таблиці мат. сподівань вартостей пошуку в піддеревах, сум ймовірностей пошуків у піддеревах та коренів піддерев відповідно:
3. Отримав дерево:

Отримане дерево в точності повторює результат з посібника ([1], ст. 427).

**Література**

* Кормен, Лейзерсон, Рівест, Штайн. Алгоритми: побудова і аналіз, 2-е видання. – 2005