ChatGPT

Київський національний університет імені Тараса Шевченка Факультет комп’ютерних наук та кібернетики Кафедра інтелектуальних програмних систем

Алгоритми та складність

Лабораторна робота №1 «Ідеальне хешування (рядки)»

Виконав студент 2-го курсу Групи ІПС-21

[Прізвище Ім’я По-батькові]

2025

Завдання Реалізувати ідеальне хешування для статичної множини рядків з алфавітним порядком. Використати авторські структури даних (без STL-контейнерів).

Теорія Ідеальна хеш-функція (perfect hash function) відображає кожний унікальний ключ на унікальне хеш-значення, тобто гарантує відсутність колізій. Для статичної множини S можна побудувати дворівневу структуру: • перший рівень розподіляє ключі по m кошиках;
• кожен кошик отримує власну хеш-функцію другого рівня, що дає ін’єкцію у підтаблицю розміру k^2 (k – розмір кошика).

Предметна область Тип даних: рядки латиницею (ASCII), що мають лінійний порядок (словниковий).
Особливості: довжина слова $\leq$ 32 символи; алфавітний порядок важливий лише для виведення, а не для хешування.

Алгоритм побудови 1. **Перший рівень.** Фіксована універсальна функція $h_1(x) = ((a_1 \cdot g(x)+b_1) \bmod P) \bmod m$, де g(x) – поліноміальний хеш рядка. Ключі розкладаються у buckets[i]. 2. **Сортування кошиків.** Для покращення середньої кількості спроб сортуємо кошики за спаданням k. 3. **Другий рівень.** Для кожного кошика перебираємо пари (a,b) доти, поки $h_2(x)=((a \cdot g(x)+b) \bmod P) \bmod k^2$ не дає повторів.
4. Зберігаємо (a,b) і підтаблицю. Якщо k=1, достатньо зберігти індекс.

Пошук у таблиці 1. h_1 визначає кошик j.
2. Якщо rows[j].size = 0 → елемент відсутній.
3. Інакше h_2 визначає позицію idx у підтаблиці rows[j].table.
4. Перевіряємо, чи rows[j].table[idx] = шукана строка.

Складність • Побудова – O(n) очікувано;
• Пошук – O(1) у найгіршому випадку;
• Пам’ять – O(n).

Мова програмування C++20 (компілятор – g++-13). STL-контейнери замінено на примітивні динамічні масиви.

Модулі програми 1. **StringHash** – поліноміальний хеш (BASE=257, mod=P). 2. **Bucket** – простий динамічний масив рядків (push, деструктор). 3. **HashRow** – зберігає (a,b), size=k^2 та підтаблицю std::string. *4. PerfectHashTable – методи build(), contains(), print(). 5. main** – демонстраційний приклад.

Інтерфейс користувача Користувач не вводить дані вручну; демонстраційний набір з 10 слів заданий у коді. При бажанні можна зчитати набір із файлу.

Тестовий приклад Множина слів: { apple, banana, grape, kiwi, lemon, mango, orange, peach, plum, watermelon } Параметри:
• n = 10 → m = 10;
• P = 1 000 019;
• a_1 = 31, b_1 = 17.
Далі розрахунок h_1 та розподіл по кошиках (див. консольний вивід програми). Більшість кошиків містять 1 елемент, тому h_2 підбирається миттєво.

Висновки Отримано ідеальну хеш-таблицю для статичної множини рядків. Структура забезпечує O(1) доступ без колізій та має передбачуване споживання пам’яті. Основне обмеження – таблиця є статичною: додавання нового слова потребує повної перебудови.

Література 1. CLRS, 4-те вид.; розд. 11.5.
2. https://en.wikipedia.org/wiki/Perfect_hash_function
3. І. Дрозд, «Алгоритми й структури даних». Розд. 10.