

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)»

Варіант 20. Тип даних T1

Виконав студент 2-го курсу
Групи ІПС-21

Шевнюк Михайло Олексійович

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 ≤ 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 зберегти індекс.

Пошук у таблиці

1. h_1 визначає кошик j .
2. Якщо `rows[j].size = 0` \rightarrow 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++

Модулі програми

1. **StringHash** – поліноміальний хеш ($\text{BASE}=257$, $\text{mod}=P$).
2. **Bucket** – простий динамічний масив рядків (push, деструктор).
3. **HashRow** – зберігає (a,b) , $\text{size}=k^2$ 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:

- $n = 10 \rightarrow m = 10$;
- $P = 1\,000\,019$;
- $a_1 = 31$, $b_1 = 17$.

Далі розрахунок h_1 та розподіл по кошиках (див. консольний вивід програми). Більшість кошиків містять 1 елемент, тому h_2 підбирається миттєво.

```

// -----
// Демонстрація
// -----
int main() {
    const string words[] = {
        "apple", "banana", "grape", "kiwi", "lemon",
        "mango", "orange", "peach", "plum", "watermelon"
    };
    const int N = size(words);

    PerfectHashTable ht;
    ht.build(words, N);

    ht.print();

    cout << boolalpha;
    cout << "contains(\"kiwi\")"    = " << ht.contains(s: "kiwi")    << '\n';
    cout << "contains(\"papaya\")" = " << ht.contains(s: "papaya") << '\n';
}

```

Результат

== bucket 0 ==

a=1230143 b=9784496 size=1

[0] watermelon

== bucket 1 ==

a=4847307 b=2377292 size=1

[0] mango

== bucket 2 ==

a=531633 b=2667117 size=1

[0] grape

== bucket 3 ==

a=6598359 b=4926094 size=9

[3] apple

[6] peach

[7] kiwi

== bucket 5 ==

a=1461665 b=7714908 size=1

[0] lemon

== bucket 7 ==

a=1206847 b=6273623 size=1

[0] orange

== bucket 8 ==

a=4486 b=1670954 size=1

[0] plum

== bucket 9 ==

a=7486846 b=1491801 size=1

[0] banana

contains("kiwi") = true

contains("papaya") = false

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.