Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Алгоритми та складність

Завдання №1

“Ідеальне хешування”

Виконав студент 2-го курсу

Групи К-29

Пащенко Дмитро Вікторович

2020

**Завдання**:

Реалізувати ідеальне хешування, 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:**

* Хеш-таблиця є узагальненням звичайного масиву: замість безпосереднього використання ключа в якості індексу масиву, індекс підраховується за значенням ключа.
* Хеш-функція: h: U→{0, 1, …, m - 1}, де U – простір ключів, область прибуття – комірки хеш-таблиці Т[0..m-1]. Елемент з ключем k хешується в комірку h(k).
* Універсальна хеш-функція – це така хеш-функція, що ймовірність виникнення колізії між двома різними ключами не перевищує ймовірності співпадіння двох випадково обраних хеш-значень із множини {0, 1, …, m - 1}, яка рівна 1/m.
* Визначимо хеш-функцію ha,b для будь-яких a є {1, …, p - 1} та b є {0, …, p - 1}, де p – таке достатньо велике просте число, що всі ключі знаходяться в проміжку від 0 до p – 1 включно, таким чином:  
  ha,b(k) = ((a\*k + b) mod p) mod m (\*). Така множина хеш-функцій є універсальною.
* Ідеальне хешування – це двохрівневе хешування, що використовує ідеальне хешування на кожному рівні. Цей метод виконує пошук елемента за О(1) звернень до пам’яті в найгіршому випадку.
* На першому рівні n ключів хешуються в m-комірок з використанням підібраної універсальної хеш-функції h типу (\*). Однак замість хешування списку ключів в j-ту комірку ми створюємо вторинну хеш-таблицю Sj зі своєю універсальною хеш-функцією hj типу (\*), яка підбирається випадковим чином, доки не зникнуть усі колізії.
* Для забезпечення відсутності колізій кількість комірок вторинної хеш-таблиці mj має дорівнювати квадрату кількості ключів, що в неї хешуються nj.

**Мова програмування:** С++.

**Інтерфейс користувача**

Введення даних відбувається через консоль.

Вхідні дані: попередньо заданий та захешований масив значень (у даному випадку масив станцій), s – ім’я шуканої станції.

Повертає: ім’я шуканої станції, знайдене шляхом її пошуку за ключем в ідеальній хеш-таблиці.

**Модулі програми:**

* **long long compute\_hash(std::string const &str)**

Вираховує ключ для рядка str.

**Складність:** , m – кількість символів у рядку.

* **long long compute\_hash(station const &station)**

Вираховує ключ для компанії company.

**Складність:** , m – кількість символів в імені станції.

* **unsigned secondary\_hash\_table::get\_index(long long const &key) const**

Метод, що вираховує індекс ключа за хеш-функцією (\*) із встановленими в класі коефіцієнтами.

**Складність:**

* **secondary\_hash\_table::secondary\_hash\_table(std::vector<T> const &data, long long (\*\_hash\_func)(T const &))**

Конструктор, що формує вторинну хеш-таблицю шляхом випадкового перебору коефіцієнтів функції (\*), доки не зникнуть колізії. Приймає масив значень, що потрапили в цю таблицю, та покажчик на функцію хешування для шаблонного параметра Т.

* **T secondary\_hash\_table::get(long long const &key) const**

Повертає значення елемента з ключем key з вторинної таблиці.

**Складність:** О(1).

* **perfect\_hash\_table::perfect\_hash\_table(std::vector<T> const &data, long long (\*\_hash\_func)(T const &))**

Конструктор, що формує первинну хеш-таблицю та розподіляє ключі по вторинним хеш-таблицям. Приймає масив значень, які потрібно захешувати, та покажчик на функцію хешування для шаблонного параметра Т.

* **T perfect\_hash\_table::get(long long const &key) const**

Повертає значення елемента з ключем key, звертаючись спочатку до відповідної комірки первинної хеш-таблиці, а згодом здійснює пошук по вторинній таблиці.

**Складність:** О(1).

**Тестові приклади**

Склав список з декількох станцій, здійснив їх пошук. Перевірив співпадіння вхідного та вихідного списків.

Изображение выглядит как текст

Автоматически созданное описание

**Література**

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