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

Алгоритми та складність

Лабораторна робота №2

Завдання №1 – Ідеальне хешування

Тип даних – рядки

Виконав студент 2-го курсу
Групи ІПС-21
Павлюченко Василь

2023

Зміст

Теоретичні відомості

Хешування – процес прив’язки вхідних даних будь-яких розмірів до фіксованих розмірів, що є «відбитком» вхідних даних. Цей процес є одностороннім, тобто з хеш-значення неможливо відновити початкові дані. Функція, яка виконує процес хешування називається **хеш-функцією**.

Універсальна хеш-функція – хеш-функція, яка випадковим чином обирається з сімейства хеш-функцій з певною математичною властивістю. Від цього очікується низька кількість *колізій*.

Хеш-таблиця – ефективна структура даних, яка використовується для реалізації асоціативних масивів (словників), який є абстрактним типом даних, що прив’язує ключі до певних значень в кошиках.

В ідеальному варіанті, хеш-функція прив’язує кожен ключ до унікального кошика, але більшість хеш-таблиць використовують неідеальні хеш-функції, які можуть призводити до *колізій*.

Колізія – випадок, коли хеш-функція генерує однакове хеш-значення для більш ніж одного ключа.

Щоб зменшити вірогідність виникнення колізій є декілька методів, одним з яких є використання *ідеального хешування*.

Ідеальне хешування – хеш-функція, яка прив’язує різні елементи з множини ключів S до множини m цілих чисел без колізій. Мовою математичних термінів, це є *ін’єктивною функцією*. Важливим є те, що множина ключів – статична, тобто не змінюється після збереження в таблицю.

Алгоритм

Ідеальне хешування складається з двох рівнів.

Перший рівень хешування: n ключів хешуються в m комірок за допомогою універсальної хеш-функції h. Так як у моєму варіанті тип даних для хешування це рядки, то хеш-функція має наступний вигляд:

$$hashValue = (hashValue * a + c[i])\% \, tableSize,$$

де *hashValue* – хеш-значення поточного ключа, *a* – параметр, який обирається випадковим чином з діапазону [1, 100], *c[i]* – поточний символ рядку, *tableSize* – розмір хеш-таблиці.

У хеш-функцію передається рядок (ключ), для якого посимвольно обчислюється його хеш-значення. Використання параметру *a* робить дану хеш-функцію *універсальною*.

Другий рівень хешування: для кожної комірки, яка має колізії, створюється своя вторинна хеш-таблиця зі своєю універсальною хеш-функцією, вибраною так, щоб уникнути колізій; розмір цієї вторинної хеш-таблиці – квадрат кількості ключів, захешованих в комірку.

Разом з кожною вторинною хеш-таблицею зберігається свій масив коефіцієнтів, який складається з двох значень. Перше значення – розмір вторинної хеш-таблиці. Друге значення – параметр *a*, який використовувався для універсальної хеш-функції відповідної вторинної хеш-таблиці.

Для знаходження ключа в хеш-таблиці використовуються хеш-значення, отримані з відповідними параметрами *a*. Саме для цього вони зберігаються в масиві.

Для уникнення помилок, перед початком хешування масив ключів перевіряється на повторні значення, які потім видаляються, якщо були знайдені.

Складність

Складність першого рівня хешування складає *O(n)*, так як ми проходимо кожен ключ у масиві лише один раз.

Складність другого рівня хешування складає $O(n^2)$, так як для кожного ключа який ми хешуємо, ми можемо мати колізію, і в такому випадку доводиться повторно хешувати всі ключі в даному кошику.

Складність доступу до елементу складає O(1), так як операція виконується за константний час.

Мова програмування

C++

Модулі програми

Використані класи з Стандартної Бібліотеки C++:

```
#include <iostream>
#include <string>
#include <vector>
#include <random>
```

Типи даних:

```
typedef unsigned int uint;
typedef vector<vector<string>> hashTable;
```

Функції:

```
// Універсальна хеш-функція для рядків
uint hash(string& key, uint tableSize, uint a);

// Видалення повторень у масиві рядків
void removeRepetitions(vector<string>& array);

// Перший рівень хешування: масив ключів хешується за допомогою
універсальної хеш-функції
hashTable firstLevelHash(vector<string>& array,
vector<vector<uint>>& coefficients, uint a);

// Другий рівень хешування: для кожної комірки своя вторинна
хеш-таблиця зі своєю універсальною хеш-функцією
void secondLevelHash(hashTable& table, vector<vector<uint>>&
coefficients);

// Ідеальне хешування для масиву рядків
hashTable perfectHash(vector<string>& array,
vector<vector<uint>>& coefficients, uint a);

// Пошук ключа у хеш-таблиці
void findKey(string& key, hashTable& table,
vector<vector<uint>>& coefficients, uint a);

// Вивід хеш-таблиці в консоль
void printTable(hashTable& table, vector<vector<uint>>&
coefficients);
```

Демонстрація роботи

Вхідні дані:

```
vector<string> array = {
        "apple", "banana", "cherry", "date", "elderberry",
        "fig", "grape", "honeydew", "ice-cream", "jackfruit",
        "kiwi", "lemon", "mango", "nectarine", "orange"
};
```

Відображення хеш-таблиці:

```
Menu:
1. Print hash table
2. Find key
3. Exit
Enter your choice:1
 i|   Coefficients      | Values
---------------------------------------
0|      0 0             |
1|      4 47            | kiwi - orange -
2|      0 0             |
3|      4 139           | - cherry - nectarine
4|      1 0             | lemon
5|      16 986          | - - - - - mango - - - fig - ice-cream - apple - -
6|      0 0             |
7|      9 478           | - elderberry - honeydew - grape - - -
8|      1 0             | jackfruit
9|      4 123           | - banana date -
```

Пошук неіснуючого ключа:

```
Menu:
1. Print hash table
2. Find key
3. Exit
Enter your choice:2
 Enter key to find:coconut
 Hash values are: 8 and 0
coconut not found.
```

Пошук існуючого ключа:

```
Menu:
1. Print hash table
2. Find key
3. Exit
Enter your choice:2
 Enter key to find:grape
 Hash values are: 7 and 5
grape found.
```

Інтерфейс користувача:

Текстове меню у консолі, де користувач може відобразити хеш-таблицю або знайти ключ.

Висновки

Ідеальне хешування є ефективним способом організації даних, який забезпечує швидкий доступ до ключів. Однак, воно має деякі недоліки, такі як великий обсяг пам’яті, необхідний для зберігання вторинних хеш-таблиць і масиву коефіцієнтів, а також складність вибору оптимальних параметрів *a* для універсальних хеш-функцій. Тому, для практичного застосування ідеального хешування потрібно враховувати ці фактори.

Використані літературні джерела:

- **https://en.wikipedia.org/wiki/Hash_function**
- **https://en.wikipedia.org/wiki/Hash_table**
- **https://en.wikipedia.org/wiki/Hash_collision**
- **https://en.wikipedia.org/wiki/Perfect_hash_function**
- **https://en.wikipedia.org/wiki/Universal_hashing**
- **Cormen T., Leiserson C., Riverst R., Stein C. – Introduction to Algorithms, 4th Edition – 2022, [11 Розділ]**