Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних інформацій систем

Алгоритми та складність

**Лабораторна робота №1**

Завдання №1 – Ідеальне хешування

Тип даних – раціональні числа

Виконав студент 2-го курсу

Групи ПІ-22

Кіщук Ярослав

2023

**Зміст**

[Теоретичні відомості 3](#_heading=h.gjdgxs)

[Алгоритм 4](#_heading=h.30j0zll)

[Складність 6](#_heading=h.1fob9te)

[Мова програмування 7](#_heading=h.3znysh7)

[Модулі програми 7](#_heading=h.2et92p0)

[Інтерфейс користувача 9](#_heading=h.tyjcwt)

[Приклади 9](#_heading=h.3dy6vkm)

[Висновок 12](#_heading=h.1t3h5sf)

[Література 12](#_heading=h.4d34og8)

# **Теоретичні відомості**

**Ідеальна хеш-функція** – Ідеальна хеш-функція - це хеш-функція, яка гарантує відсутність колізій. Тобто, кожен ключ має свій унікальний хеш-код і кожен елемент може бути знайдений безпосередньо за його хеш-кодом. Ідеальну хеш-функцію можна використовувати для створення ідеальної хеш-таблиці, яка забезпечує швидкий доступ до даних без необхідності здійснювати пошук в основній таблиці..

**Множина ключів статична** – множина ключів, які використовуються для створення хеш-таблиці, залишається незмінною після того, як дані були збережені в таблицю. З точки зору математики, це означає, що хеш-функція є повною ін'єкцією, тобто кожен ключ має унікальний хеш-код і не існує два ключі, які матимуть однаковий хеш-код.

**XOR** (ексклюзивне або) - це логічна операція, яка повертає істину (1), якщо один з операндів має значення істини, а другий операнд має значення хиби, інакше вона повертає хибу (0).

Таблиця істинності для XOR:

|  |  |  |
| --- | --- | --- |
| A | B | A XOR B |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**big.Rat –** структура, яка представляє собою частку a/b (дріб) довільної точності. Нульовим значенням для big.Ratє значення 0.

Представлення на мові Golang:

type big.Rat struct {

a, b Int

}

# **Алгоритм**

1. Отримуємо набір вхідних даних, у моєму випадку – зчитуємо раціональні числа (у вигляді дробів) з текстового файлу.
2. Перевіряємо набір вхідних даних на наявність повторів, вилучаємо їх
3. Створюємо хеш-таблицю, розмір якої дорівнює кількості елементів вхідного набору даних.
4. Потім, за допомогою хеш-функції, для кожного з елементів рахуємо його хеш-індекс у таблиці. Формула хеш-функції наведена нижче:

XOR - це логічна операція, що виконується над двома бітами (або бітовими послідовностями), і повертає значення 1, коли значення бітів різні, і 0, коли вони однакові.

Ця хеш-функція обчислює хеш-код для об'єкту типу big.Rat, використовуючи його чисельник і знаменник. Спочатку чисельник перетворюється на беззнакове 64-бітне ціле число, і, якщо чисельник від'ємний, то виконується операція побітового NOT, щоб змінити всі біти числа на протилежні. Далі, чисельник і знаменник додаються до хеш-коду за допомогою множників B і A відповідно. Хеш-код обчислюється за допомогою двох раундів операцій побітового XOR та зсуву вправо. Нарешті, хеш-код залишається в діапазоні від 0 до M-1 за допомогою операції залишку від ділення.

1. Якщо виникає ситуація, коли два або більше ключів мають однаковий хеш-код та потрапляють в одну і ту ж комірку основної хеш-таблиці, ми створюємо додаткову хеш-таблицю у цій комірці. Розмір цієї додаткової таблиці дорівнює квадрату кількості елементів, які потрапили до цієї комірки в основній таблиці.
2. Після створення додаткової хеш-таблиці, ми хешуємо кожен елемент в цій таблиці, вибравши нові значення змінних A та B. Цей процес повторюється доти, поки не будуть вирішені всі колізії та кожен елемент буде унікальним в межах додаткової таблиці.

# **Складність**

**Складність створення первинної таблиці**

Найкраща та найгірша складність - O(n2) ( n - кількість елементів у вхідному файлі), оскільки найбільш дорогою операцією є пошук повторів у початковому наборі даних.

**Складність пошуку**

Складність пошуку користувачем в даному випадку - O(1), оскільки знаходження елемента в таблиці займає постійний час. У випадку використання підтаблиць без колізій складність також залишається O(1), оскільки відсутня необхідність проходитися по всіх елементах підтаблиці.

# **Мова програмування**

**Go**

# **Модулі програми**

Основні функції:

*//Функція main є вхідним пунктом в програму і містить код для*

*//створення та взаємодії з хеш-таблицею.*

*func main()*

*// Функція перевірки числа на простоту*

*func isPrime(n int) bool*

*// Функція пошуку наступного простого числа*

*func nextPrime(n int) int*

*// Функція створює новий об'єкт типу HashFunction, який представляє*

*// хеш-функцію для таблиці з заданою кількістю слотів M.*

*func NewHashFunction(M int) \*HashFunction*

*// Функція створює нову хеш-таблицю на основі даних, збережених у файлі з вказаним ім'ям.*

*func NewHashTable(filename string) (\*HashTable, error)*

*// Ця функція призначена для читання раціональних чисел з файлу і повернення їх у вигляді масиву []\*big.Rat (раціональних чисел).*

*func ReadRats(filename string) ([]\*big.Rat, error)*

Класи та їх функції:

// Структура хеш-функції

type HashFunction struct

// Повертає хеш для заданого раціонального числа

func (hf \*HashFunction) hash(rat \*big.Rat) int

// Структура хеш-таблиці

type HashTable struct

// Функція, яка генерує підтаблицю на основі попередньої, додаючи до неї новий елемент

func (\*HashTable) NewSubTable(toAdd \*big.Rat, oldSubTable \*HashTable) \*HashTable

*// Функція, яка видалаяє однакові раціональні числа*

*func RemoveDuplicates(rats []\*big.Rat) []\*big.Rat*

*// Функція, яка перевіряє чи є раціональне число у таблиці.*

*func (table \*HashTable) Check(rat \*big.Rat) \*big.Rat*

// Функція, яка перевіряє чи є раціональне число у таблиці.

func (table \*HashTable) Check(rat \*big.Rat) \*big.Rat

*// Вивід усієї хеш-таблиці*

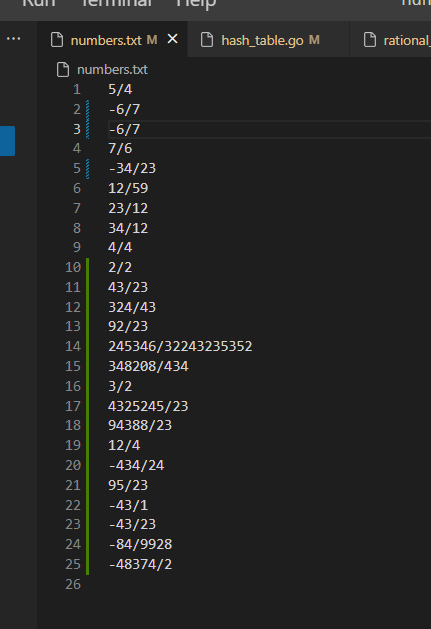
*func (ht \*HashTable) Print()*

# **Інтерфейс користувача**

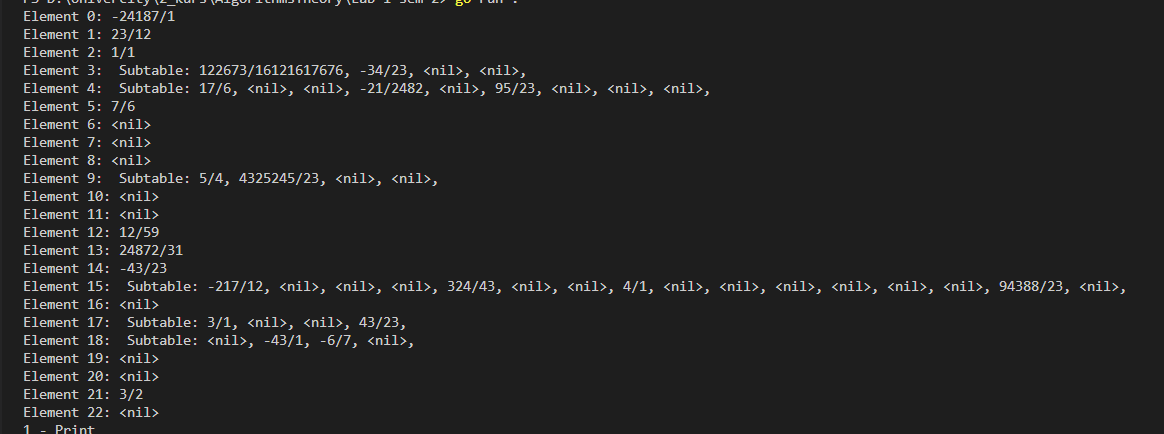
Вхідні дані задаються в файлі, а результат виводиться у консоль.

# **Приклади**

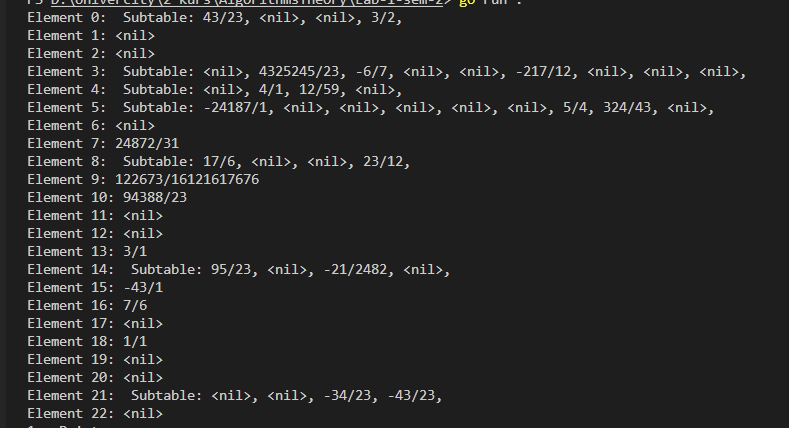
Вхідний файл



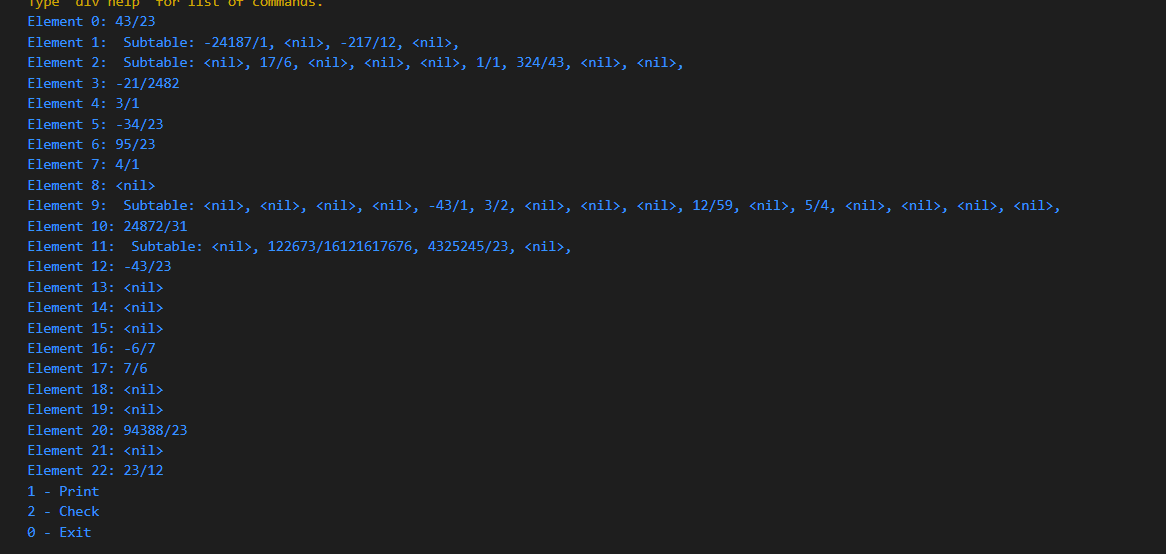
Перший запуск



Другий запуск



Третій запуск



Після того, як ми запустили програму тричі, ми отримали результати, які були показані вище. Ми зосередимо увагу на третьому випадку. Ми можемо взяти будь-який елемент з вектора, оскільки він складається з 23 елементів, для аналізу роботи нашої хеш-функції. На цей раз ми розглянемо десятий (або одинадцятий) елемент, який є дробом 24872/31 і не має колізій.

З коду, отримуємо значення випадкових змінних A = 777269 та B = 919511, M = 23, P = 29. Знаючи ці значення, обчислимо хеш-індекс.

= 2 = 113,064,390,808

= 87, 881,445,978,943,330

Ми отримали значення 10, що співпадає з номером елементу, яким є раціональне число 24872/31 (див. третє фото), що підтверджує правильність роботи нашої програми.

# **Висновок**

Ідеальне хешування - це ефективний спосіб зберігання та пошуку даних, який застосовують в обчислювальній техніці та інформаційних системах. Незважаючи на те, що його використовують рідко, алгоритм досить простий та легко реалізовується в програмному коді з лінійною складністю. Проте, варто мати на увазі, що для різних типів даних, необхідно реалізувати різні хеш-функції, щоб досягти оптимальної продуктивності та точності пошуку.

# **Література**

1. Лекція з предмету «Алгоритми та складність 1»
2. https://www.tutorialspoint.com/Hash-Functions-and-Hash-Tables
3. https://en.wikipedia.org/wiki/Perfect\_hash\_function
4. https://link.springer.com/chapter/10.1007/978-3-642-04128-0\_61
5. http://www.cs.otago.ac.nz/cosc242/pdf/L11.pdf
6. https://www.kindsonthegenius.com/perfect-hashing/