**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

**ІМЕНІ ТАРАСА ШЕВЧЕНКА**

ЛАБОРАТОРНА РОБОТА №1

З курсу “Алгоритмі і складність”

Ідеальне хешування

Виконав

Студент групи ПІ-22

Факультету комп’ютерних наук

та кібернетики

Мандзюк Дмитро

Київ 2023

**Зміст**

[Завдання 3](#_Toc128161844)

[Теорія 3](#_Toc128161845)

[Алгоритм 3](#_Toc128161846)

[Складність 4](#_Toc128161847)

[Мова програмування 4](#_Toc128161848)

[Модулі програми 4](#_Toc128161849)

[Інтерфейс користувача 5](#_Toc128161850)

[Приклади 5](#_Toc128161851)

[Висновок 7](#_Toc128161852)

[Література 7](#_Toc128161853)

# **Завдання**

Реалізувати ідеальне хешування для **дійсних чисел**.

# **Теорія**

**Ідеальна хеш-функція** – хеш-функція, яка перетворює завчасно відому статичну множину ключів в діапазон цілих чисел [0, n-1] без колізій, тобто один ключ відповідає тільки одному унікальному значенню.

**Множина ключів статична** – не змінюється після збереження в таблицю.

З точки зору математики, це повна ін’єкційна функція.

# **Алгоритм**

1. Отримуємо набір даних (ключів). В нашому випадку – набір масивів дійсних чисел.
2. Перевіряємо набір вхідних даних на наявність повторів чи пустих масивів і видаляємо їх у разі наявності.
3. Створюємо хеш-таблицю, розмір якої дорівнює кількості масивів вхідного набору даних.
4. Потім, за допомогою хеш-функції, для кожного з елементів рахуємо його хеш-індекс у таблиці. Формула перетворення дійсного числа i в натуральне наведена нижче:

де

Тоді отримуємо наступну хеш-формулу для масиву дійсних чисел:

де

1. У випадку виникнення колізії, створюємо під хеш-таблицю у комірці, у якій виникла колізія, розмір якої дорівнює квадрату кількості масивів, що потрапили в одну й ту саму комірку.
2. Після цього хешуємо кожний масив під хеш-таблиці попередньо обравши нові змінні a, b, c, k. Повторюємо цей крок до тих пір, поки не буде нових колізій.

# **Складність**

Складність створення первинної таблиці у найгіршому випадку - O(n)

Складність пошуку елемента користувачем у найгіршому випадку - O(1)

# **Мова програмування**

С++

# **Модулі програми**

Основні функції:

**int** main()

*//надходження вхідних даних і створення*

**bool** is\_prime(**int** value)

*//перевіряє чи число просте*

**int** next\_prime(**int** value)

*//знаходить наступне просте число*

**int**\* decompose\_real\_number(**double** value)

*//розкладає дійсне число в [ціла\_частина, дробова\_частина] з точністю 10^3*

*//приклад: -17,84368 -> [17, 843]*

**int** double\_to\_int\_function (**double** value, **int** p**, Constants** constants)

*//перетворює дійсне число в натуральне*

**int** hash\_function(**std::vector<double>** double\_array, **int** size, **int** p, **Constants** constants)

*//хешує масив дійсних чисел і повертає хеш-індекс*

Класи та їх функції:

**struct** HashTable

*//структура для симуляції хеш-таблиці*

**struct** Constants

*//структура для зберігання констант a, b, c з нашого алгоритму*

**void** Constants::randomize(int p)

*//рандомізує константи від 0 до p*

**HashTable(std::vector<std::vector<double>>** keys**)**

*//конструктор хеш-таблиці для вхідного массиву масивів дійсних чисел key\_array*

**void** HashTable::rehash()

*//рандомізує константи у комірках, де більше одного масива, доки коллізії не зникнуть*

**void** HashTable::print()

*//виводить хеш-таблицю у консоль*

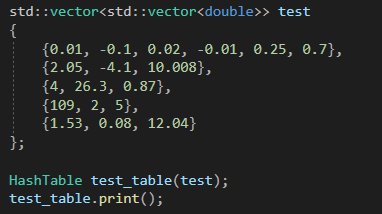
**void** HashTable::find(**std::vector<double>** key)

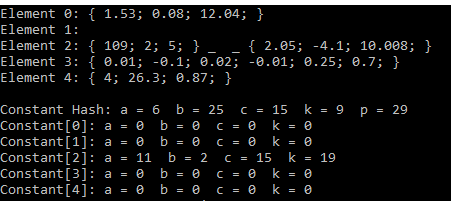
*//виводить інформацію про комірку в хеш-таблиці, де знаходиться масив дійсних чисел key*

# **Інтерфейс користувача**

Вхідні данні задаються в коді програми (а саме в наведеній вище функції int(main)), а результат виводиться у консоль.

# **Приклади**





В прикладі маємо вектор рядків з 5 масивів. Розглянемо роботу хеш-функції на масиві {2.05; -4.1; 10.008}.

Отримуємо числа a = 6, b = 25, c = 15, k = 9; p = 29 і за формулою

переводимо дійсні числа по формулі і отримуємо

Далі використовуємо хеш-формулу:

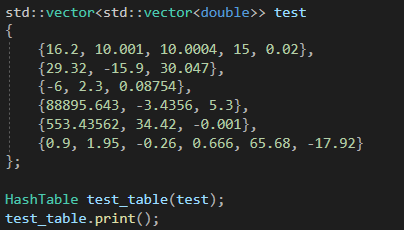
Отримуємо:

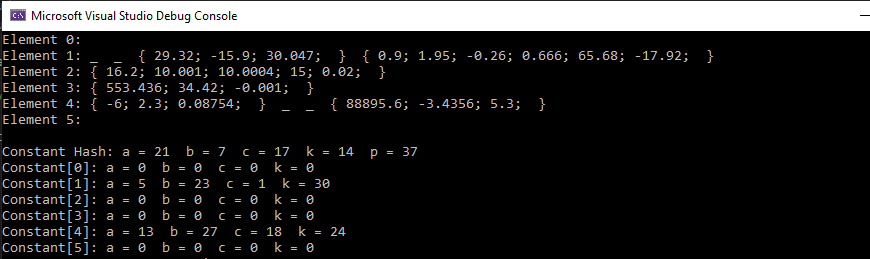
Тобто індекс комірки в головній хеш-таблиці = 2.

Далі аналогічно рахуємо допоміжний індекс на випадок колізій і отримуємо 3.

Отже індекси для масива {2.05; -4.1; 10.008} = {2; 3}. Що і відповідає дійсності.

Ще один приклад:





# **Висновок**

Ідеальне хешування – досить специфічне. Особливо у випадку використання дійсних чисел (а саме масивів дійсних чисел) в якості для ключів. В цілому краще вибирати інші типи даних для хешування.   
Переваги даного методу – очевидні. Знаходження даних за ключем в хеш-таблиці відбувається за константний час у найгіршому випадку. Але недоліки також дуже значущі – якщо у нас великі вхідні масиві, то навіть якщо наша хеш функція зможе рівномірно розподілити їх, об’єм зайнятої пам’яті у випадку колізій буде зростати дуже швидко.

# **Література**

1. Лекція з предмету «Алгоритми та складність 1»
2. http://www.cs.otago.ac.nz/cosc242/pdf/L11.pdf
3. https://en.wikipedia.org/wiki/Perfect\_hash\_function