Київський національний університет імені Тараса Шевченка

Факультет комп’ютерних наук та кібернетики

Кафедра інтелектуальних інформацій систем

Алгоритми та складніcть

Лабораторна робота №1

Завдання №1 – Ідеальне хешування

Тип даних – Комплексні числа

Виконала студентка 2-го курсу

Групи ІПС - 22

Мандріченко Ксенія

2023

**Зміст**

[Теоретичні відомості 3](#_Toc102339827)

[Алгоритм 3](#_Toc102339828)

[Складність 5](#_Toc102339829)

[Мова програмування 5](#_Toc102339830)

[Модулі програми 6](#_Toc102339831)

[Інтерфейс користувача 7](#_Toc102339832)

[Приклади 7](#_Toc102339833)

[Висновок 8](#_Toc102339834)

[Література 8](#_Toc102339835)

# **Теоретичні відомості**

**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.

Ідеальне хешування має на меті повністю усунути колізії за допомогою хеш-функції, яка відображає кожен ключ на унікальний індекс у хеш-таблиці. Це досягається шляхом ретельного вибору хеш-функції на основі набору ключів, які будуть зберігатися в хеш-таблиці. Якщо ідеальна хеш-функція відображає кожен ключ на унікальний індекс із високою ймовірністю, це означає, що колізії рідкісні, а хеш-таблиця має високий ступінь ефективності та продуктивності.

# **Алгоритм**

1. Вводимо кількість векторів
2. Вводимо кількість комплексних чисел у векторах
3. Вводимо дійсну та уявну частини кожного комплексного числа та зберегаємо комплексні числа у векторі.
4. За допомогою хеш-функції, для вектора комплексних чисел рахуємо

його хеш-значення . Хеш-значення обчислюється шляхом перебору кожного елемента вектора, виконуючи такі операції над кожним елементом:

1.Взяття дійсної частини комплексного числа та хешування його за допомогою функції hash для подвійних значень.

2.Об’єднання отриманого хеш-значення з постійним значенням (0x9e3779b9) - це довільна константа, яка зазвичай використовується в хеш-функціях, оскільки було показано, що вона створює хороші хеш-значення для широкого діапазону вхідних даних.Константа додається до хеш-значення кожної частини, щоб гарантувати, що отримані хеш-значення не зміщені щодо певних бітових шаблонів.

3.Зсув початкового значення вліво на 6 бітів, а потім вправо на 2 біти для розподілу результуючого хеш-значення, додатково зменшуючи зміщення та збільшуючи ймовірність отримання унікального хеш-значення для різних вхідних векторів

Потім побітова операція XOR «^» використовується для об'єднання хеш-значень дійсної та уявної частин, яка створює одне хеш-значення для всього комплексного числа. Та об’єднання хеш-значення комплексного числа з початковим значенням яке дорівнює розміру вектора який хешується .

(Побітова операція XOR використовується для забезпечення того, щоб невеликі зміни в дійсній або уявній частині комплексного числа призводили до значної зміни хеш-значення, що допомагає мінімізувати кількість колізій у хеш-таблиці.)

Повторення кроків 1-3 з уявною частиною комплексного числа.

Кінцеве хеш-значення є результатом усіх цих операцій, виконаних над усіма елементами вектора.

1. Далі скористаємося спеціальною функцією рівності ComplexVectorEqual, яка використовується unordered\_map для визначення рівності двох ключів (векторів комплексних чисел). Функція operator() приймає два вектори комплексних чисел vec1 і vec2 і повертає логічне значення, яке вказує, чи вони рівні чи ні.

Функція спочатку перевіряє, чи два вектори мають однаковий розмір. Якщо вони цього не роблять, він одразу повертає false, оскільки вектори не можуть бути рівними, якщо вони мають різні розміри. Якщо вектори мають однаковий розмір, функція повторює кожен елемент векторів і перевіряє, чи вони однакові. Якщо будь-який елемент vec1 не дорівнює відповідному елементу у vec2, функція повертає false. Якщо всі елементи рівні, функція повертає true, вказуючи, що два вектори рівні.

1. Створюємо хеш-таблицю за допомогою контейнеру unordered\_map використовуючи спеціальну хеш-функцію та функцію рівності, щоб визначити, як зберігати та отримувати значення на основі ключів, які в даному випадку є векторами комплексних чисел.
2. Далі виводимо вміст таблиці. Для кожної пари ключ-значення на карті він друкує вектор ключа (який є вектором комплексних чисел), а потім відповідне значення, яке є цілим числом.

**Складність**

Обчислювальна складність цього алгоритму залежить від розміру вхідних даних, тобто кількості векторів і комплексних чисел, введених користувачем.

Хеш-функція, яка використовується в цій програмі, має часову складність O(n), де n — розмір вектора комплексних чисел, що хешуються. Це означає, що час, необхідний для хешування кожного вектора, пропорційний його розміру.

Функція рівності, яка використовується в цій програмі, має часову складність O(n), де n — розмір меншого з двох векторів, які порівнюються.

Загалом, часова складність програми залежить від кількості та розміру векторів, які вводить користувач. Якщо всі вектори мають приблизно однаковий розмір, то часова складність буде домінувати через операцію хешування, яка становить O(n) для кожного вектора. Якщо вектори мають різні розміри, то часова складність функції рівності також може стати фактором.

# **Мова програмування**

С++

# 

# **Модулі програми**

**struct** ComplexVectorHash // хеш-функція для векторів комплексних чисел . Функція приймає вектор комплексних чисел як вхідні дані та повертає хеш-значення типу size\_t.

**struct** ComplexVectorEqual // функція рівності для векторів комплексних чисел , яка використовується unordered\_map для визначення рівності двох ключів (векторів комплексних чисел). Функція operator() приймає два вектори комплексних чисел vec1 і vec2 і повертає логічне значення, яке вказує, чи вони рівні чи ні.

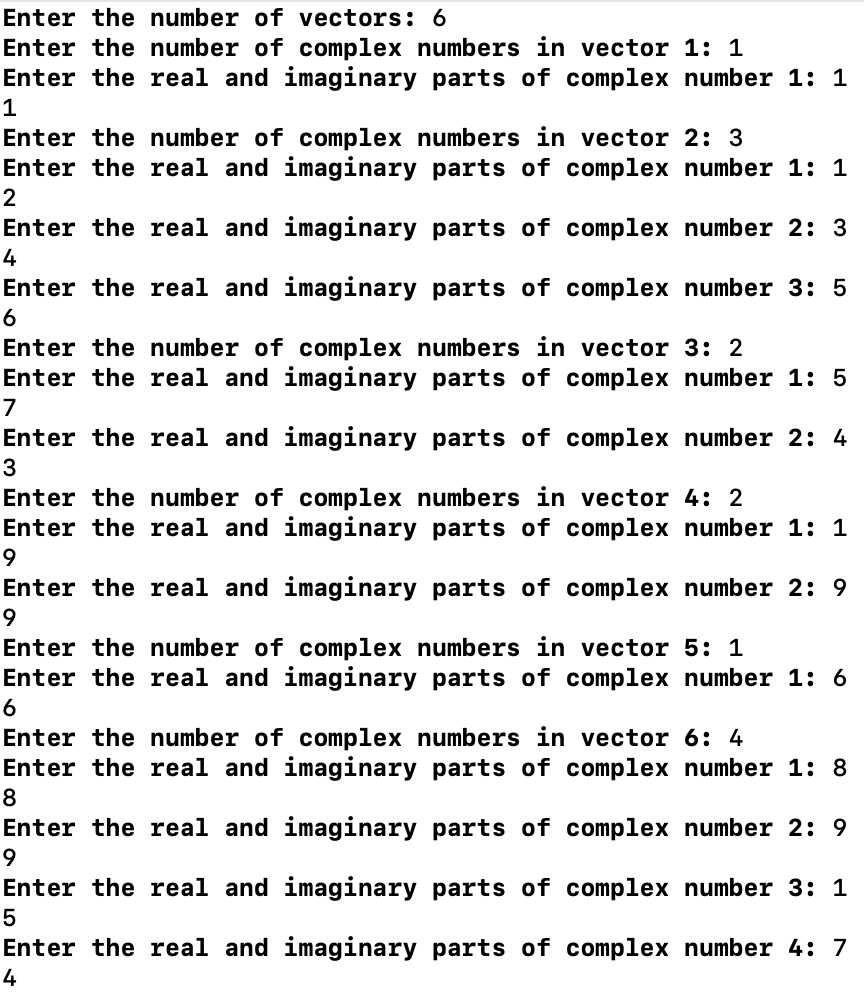
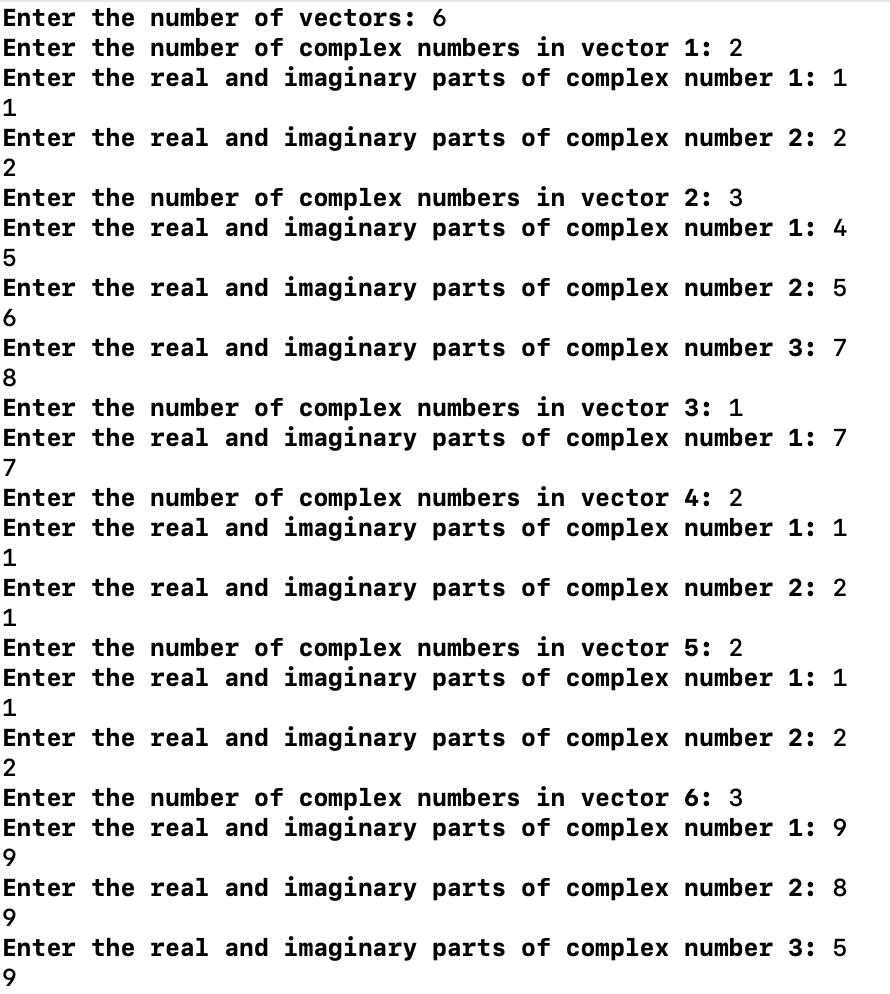
**int** main() // головна функція в якій таблиця заповнюється кількістю векторів комплексних чисел, введених користувачем. Для кожного вектора користувачеві пропонується ввести кількість комплексних чисел у векторі, а потім ввести кожне комплексне число. Потім введені вектори хешуються за допомогою функції ComplexVectorHash і зберігаються в таблиці з відповідними цілочисельними значеннями, призначеними як послідовні індекси. Нарешті, вміст таблиці виводиться на консоль разом із кожним вектором і відповідним хеш-значенням.

unordered\_map // стандартний контейнер у C++, який реалізує хеш-таблицю. unordered\_map визначає невпорядковану карту під назвою "таблиця", яка використовує вектори комплексних чисел як ключі та цілі числа як значення. Для ключів використовується хеш-функція ComplexVectorHash, та ComplexVectorEqual функція рівності. Це означає, що карта використовуватиме спеціальну хеш-функцію та функцію рівності, щоб визначити, як зберігати та отримувати значення на основі ключів, які в даному випадку є векторами комплексних чисел.

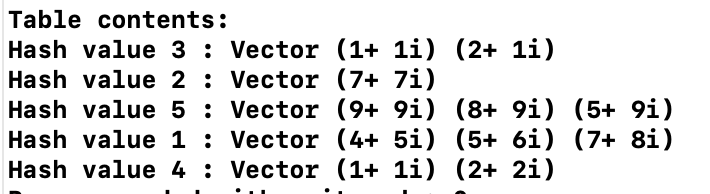
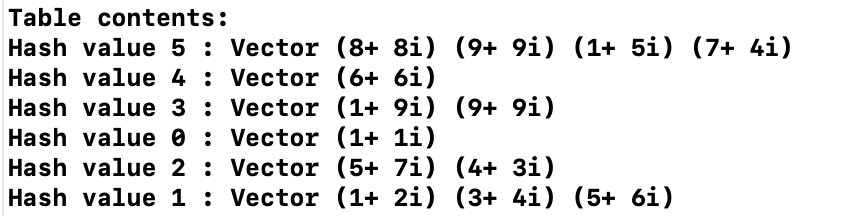
# 

# **Інтерфейс користувача**

Вхідні данні задаються з клавіатури, а результат виводиться у консоль.

# **Приклади 2)**



**1)** На основі отриманих даних ми бачимо, що програма успішно хешувала набір векторів, що містять комплексні числа. Кожному вектору присвоєно хеш-значення. Програма відображає хеш-значення, а потім вміст вектора в дужках, причому кожне комплексне число представлено як (дійсне+уявне).

Наприклад, вектор (8+8i) (9+9i) (1+5i) (7+4i) хешовано до значення 5, тоді як вектор (6+6i) хешовано до значення 4. Вектор ( 1+9i) (9+9i) хешовано до значення 3, вектор (1+1i) хешовано до значення 0, вектор (5+7i) (4+3i) хешовано до значення 2, і вектор (1+2i) (3+4i) (5+6i) хешовано до значення 1.

**2)** На основі отриманих даних ми бачимо, що програма успішно хешувала набір векторів, що містять комплексні числа. Кожному вектору присвоєно хеш-значення, а вектори з однаковим вмістом мають однакове хеш-значення. Наприклад вектор 1 та 5 однакові (1+1i)(2+2i) тому мають одне хеш-значення 4 .

# **Висновок**

Ідеальне хешування може бути корисним інструментом для хешування векторів комплексних чисел або інших типів структур даних, де традиційні хеш-таблиці можуть не забезпечити ефективну продуктивність. Він може забезпечити постійний доступ до елементів даних і допомогти зменшити кількість колізій. Однак часова та просторова складність ідеального хешування може бути вищою порівняно з традиційними хеш-таблицями, тому важливо ретельно розглянути компроміс між ефективністю та складністю алгоритму.

# **Література**

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