Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Лабораторна робота №1

“Реалізація ідеального хешування”

Виконав студент 2-го курсу

Групи ІПС-21

Дубина Андрій

2024

**Завдання**:

Реалізувати ідеальне хешування для дійсних чисел.

**Теорія**

В інформатиці ідеальна хеш-функція h для набору S — це геш-функція, яка відображає різні елементи в S на набір з m цілих чисел без зіткнень. З математичної точки зору це ін’єктивна 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**

C++20

**Модулі програми**

int main() { //основна функція, створення

struct Constants //зберігання констант для хеш-формули

struct HashTable //структура хеш-таблиці, допоміжні функції

bool is\_prime(int value) //перевірка на простоту

int next\_prime(int value) //наступне просте число після заданого значення

int\* decompose\_real\_number(double value) //"розбираємо" число в масив [ціла частина, дробова частина] з точністю 1000

int double\_to\_int\_function(double value, int p, Constants constants) //перетворення дійсного в натуральне

int hash\_function(std::vector<double> double\_array, int size, int p, Constants constants) //хешування масиву, повернення хеш-таблиці

void Constants::randomize(int p) //знаходження випадкових констант

HashTable::HashTable(std::vector<std::vector<double>> keys) //конструктор хеш-таблиці для вхідного масиву масивів keys

void HashTable::rehash() //пере-хешування для зникнення колізій, рандомізація констант

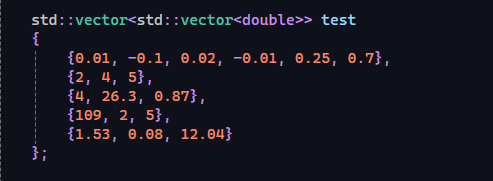
void HashTable::print() //вивід хеш-таблиці

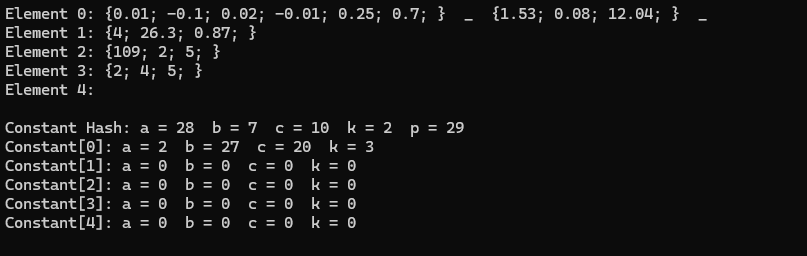
void HashTable::find(std::vector<double> key) //виводить інформацію про комірку в хеш-таблиці, де знаходиться масив дійсних чисел key

**Інтерфейс користувача**

Вхідні дані вводяться програмно (в функції int main()). Результат виводиться в консоль.

**Тестові приклади**





Розглянемо роботу програми на прикладі масиву {1.53, 0.08, 12.04}.

Маємо константи:   
a = 28, b = 7, c = 10, k = 2, p = 29

За формулою

переводимо числа.

z(1.53) = ((1\*28 + 5300\*7 + 0) mod29) = 8

z(0.08) = ((0\*28 + 80\*7 + 0)mod29) = 9

z(12.04) = ((12\*28 + 40\*7 +0)mod29) = 7

Використаємо хеш-формулу:

Індекс в головній таблиці = 0.

Аналогічно рахуємо індекс в доп. таблиці

a = 2, b = 27, c = 20, k = 3, p = 29

z(1.53) = ((1\*2 + 5300\*27 + 0) mod29) = 16

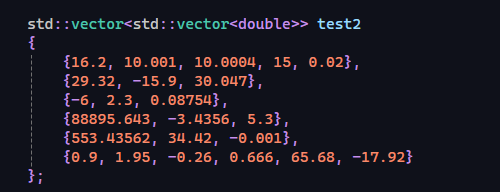
z(0.08) = ((0\*2 + 80\*27 + 0)mod29) = 14

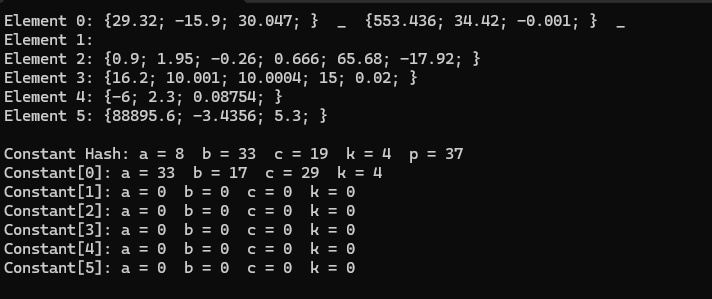
z(12.04) = ((12\*2 + 40\*27 +0)mod29) = 2

Використаємо хеш-формулу:

Індекс в допоміжній таблиці = 1.

Ще один приклад:





**Висновок**

Ідеальне хешування є ефективним методом забезпечення унікального відображення даних у хеш-таблиці. Для дійсних чисел основною викликаючою проблемою є необхідність врахування нескінченної кількості можливих значень. Успішно вирішеною була задача розробки хеш-функції, яка забезпечує ефективне розподілення значень та мінімізацію колізій.

Ідеальне хешування для дійсних чисел виявляється ефективним інструментом при обробці числових даних, де точність та швидкодія грають важливу роль у виконанні завданя, проте при великих вхідних масивах об’єм пам’яті у випадку колізії швидко зростає.

**Джерела**

1. https://en.wikipedia.org/wiki/Perfect\_hash\_function
2. https://www.cs.otago.ac.nz/cosc242/pdf/L11.pdf
3. Cormen, Thomas H.; Leiserson, Charles E.; Rivest, Ronald L. (1990). *Introduction to Algorithms* (1st ed.)