Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних інформаційних систем

Алгоритми та складність

Завдання №3

“Ідеальне хешування векторів дійсних чисел.”

Виконав студент 2-го курсу

Групи К-28

Адамов Олексій Віталійович

2021

**Завдання**

Виконати ідеальне хешування статичного масиву векторів дійсних чисел.

**Теорія**

Ідеальне хешування – хешування, яке має O(1) звертань до пам’яті у найгіршому випадку.

Нехай розмір масиву ключів – , – розмір хеш-таблиці.

Нехай – множина ключів. Сімейство хеш функцій називається універсальним, якщо .

Теорема. Для будь-якої універсальної хеш-функції з сімейства універсальних хеш функцій, якщо розмір хеш-таблиці дорівнює то ймовірність того, що не виникне жодної колізії не менше .

Ідеальне хешування відбувається у 2 кроки:

1) ключів хешуються у m комірок (причому хеш-функцію можна обрати універсальною і зробити декілька спроб хешування, щоб зменшити кількість колізій);

2) для кожної комірки створюється хеш-таблиця (її розмір - квадрат кількості ключів, захешованих в комірку на першому кроці) зі своєю унікальною універсальною хеш-функцією, вибраною так, щоб уникнути колізій.

Універсальна функція хешування для цілих невід’ємних чисел :

де – фіксоване просте число , та вибираються випадковим чином,

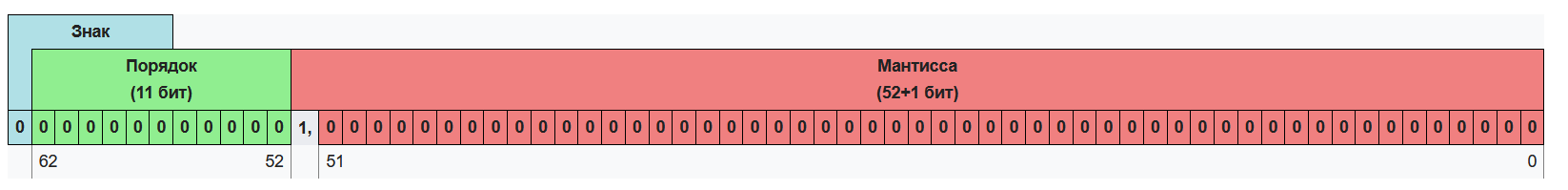
Універсальна функція хешуваннястроки (вектора невід’ємних чисел для довжини якого не відома верхня границя) довжини :

де – фіксоване просте число , вибираються випадковим чином, , - вибирається випадковим чином з сімейства універсальних.

Якщо може дорівнювати 0 краще додати якесь число, наприклад 1, до кожного . Також візьмемо попередню універсальну функцію хешування. Отримаєм:

Але за умовою задачі потрібно хешувати дійсні числа, а формули працюють тільки для цілих невід’ємних. Що робити?

Згадаймо, що у комп’ютері вся інформація представляється як послідовність бітів. Зокрема 64-бітне дійсне число (тип double у C++) представляється таким чином:



+0 і -0 будемо інтерпретувати як одне й те саме число +0.

Будемо інтерпретувати 64-бітне дійсне число як вектор з 4 16-бітних цілих невід’ємних чисел, утворених взяттям перших 16 бітів дійсного числа, другого, і так далі. Тоді вхідний вектор дійсних чисел можна інтерпретувати як такий вектор цілих невід’ємних чисел ( (замість кожного дійсного числа записали 4 невід’ємних цілих числа його представлення). До такого вектора вже можна застосувати універсальне хешування за останньою формулою.

**Алгоритм**

Побудова хеш-таблиці.

Нехай – вхідний масив ключів розміру ,

table – двовимірна хеш таблиця .

1. Вибираємо – найменьше просте число більше та кожного .

2. раз

1) Вибираємо випадкові .

2) Обчислюємо space - місце яке знадобиться для збереження ідеальної хеш-таблиці з подвійним хешуванням.

3) Запам’ятовуємо , якщо space – найменьше з знайдених на попередніх кроках циклу.

3. Використовуємо універсальну хеш-функцію з параметрами знайдених на попередньому кроці.

4. Ініціалізуємо додаткові хеш таблиці (розмір – квадрат кількості ключів які туди хешуються за ) NIL-ами. Та вибираємо параметри для - випадковим чином, якщо розмір більше 1, та покладаємо , якщо розмір = 1 (таким чином ). Якщо розмір = 0, .

5. Цикл по .

1) .

2) Якщо , записуєм туди посилання на .

3) Інакше випадковим чином знаходимо нові параметри , перехешовуємо всі значення посилання на які знаходяться у , та . Допоки виникають колізії повторюємо цей крок.

Перевірка чи є у .

1. .

2. Якщо , то , алгоритм завершуємо.

3. .

4. Якщо , то , алгоритм завершуємо.

5. Перевіряємо чи дорівнює елементу на який вказує . Якщо так, то , інакше .

**Складність**

– кількість ключів, – розмір хеш-таблиці.

Час створення - .

Час пошуку - , де – розмірність шуканого вектору, але маємо звернень до пам’яті, що підтверджує що хешування - ідеальне.

Пам’ять - .

**Мова програмування**

С++

**Модулі програми**

* void hash\_table::init(size\_t hash\_table\_size, std::vector<double>\* arr, size\_t array\_size)

Метод, який будує хеш-таблицю.

* size\_t universal\_hash\_vector\_double::hash(const std::vector<double>& vec, unsigned long long p, size\_t m)

Метод, який повертрає значення універсальної хеш функції вектора дійсних чисел (параметри зберігаються у самому класі)

* bool hash\_table::contains(std::vector<double>& value)

Метод, який перевіряє чи value є у хеш-таблиці.

**Інтерфейс користувача**

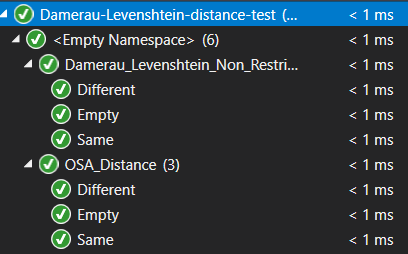
Вхідні дані вводяться з текстового файла і виводяться також в текстовий файл.

**Тестові приклади**

|  |  |
| --- | --- |
| **input.txt** | **output.txt** |
| ca abc | Damerau–Levenshtein distance (without restrictions):  2  Damerau–Levenshtein edit sequence(without restrictions):  Transpose 1 and 2  Insert after 1 b  ca->ac->abc  Has been done for 0.016 milliseconds  Optimal string alignment distance, or restricted edit distance:  3  Optimal string alignment edit sequence:  Substitute at position 2 to c  Substitute at position 1 to b  Insert after 0 a  ca->cc->bc->abc  Has been done for 0.004 milliseconds |

**Тестування**

Програма проходить тести на випадкових вхідних даних.

****

**Висновки**

Відстань Дамерау-Левенштейна є модифікацією відстані Леванштейна (додається операція транспозиції двох сусідніх символів). Обчислення обмеженої відстані потребує менше пам’яті, але потрібно пам’ятати, що обмеженої відстань не є метрикою. Відстань Дамерау-Левенштейна використовується у обробці природної мови та біоінформатиці.

**Література**

* Лекція 1 (2 семестр) з Алгоритмів та складності (Шкільняк О.С.)
* <https://en.wikipedia.org/wiki/Universal_hashing>
* [https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5\_%D1%85%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#%D0%9F%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D0%B5\_%D1%83%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE\_%D1%81%D0%B5%D0%BC%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B0\_%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B9](https://ru.wikipedia.org/wiki/%D0%A3%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D1%85%D0%B5%D1%88%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5#%D0%9F%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B5%D0%BD%D0%B8%D0%B5_%D1%83%D0%BD%D0%B8%D0%B2%D0%B5%D1%80%D1%81%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B3%D0%BE_%D1%81%D0%B5%D0%BC%D0%B5%D0%B9%D1%81%D1%82%D0%B2%D0%B0_%D1%85%D0%B5%D1%88-%D1%84%D1%83%D0%BD%D0%BA%D1%86%D0%B)
* <https://habr.com/ru/company/otus/blog/448992/>
* <https://neerc.ifmo.ru/wiki/index.php?title=%D0%9F%D1%80%D0%B5%D0%B4%D1%81%D1%82%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%B8%D0%B5_%D0%B2%D0%B5%D1%89%D0%B5%D1%81%D1%82%D0%B2%D0%B5%D0%BD%D0%BD%D1%8B%D1%85_%D1%87%D0%B8%D1%81%D0%B5%D0%BB>