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

Нехай розмір масиву ключів – .

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:

**Алгоритм**

Використаємо підхід динамічного програмування.

У будемо зберігати відстань Дамерау-Левенштейна між першими символами рядка , та першими символами рядка .

1. Обмежена відстань Дамерау-Левенштейна (optimal string alignment distance) між рядками та , і оптимальна послідовність дій для перетворення у .

1) Ініціалізуємо .

2) Ініціалізуємо .

3) Обчислюємо у двох вкладених циклах

3.1)

.

3.2) Якщо , то .

4) – шукана відстань.

5) Нехай . Будемо її перетворювати та записувати ці перетворення.

6) Починаючи з (нижній правий кут матриці ), до поки та одночасно не будуть дорівнювати 0 (верхній лівий кут матриці ), обчислюємо

. (Не враховуємо компоненту, якщо індекс < 0, хоча б один з компонентів буде визначеним). Далі

6.1) Якщо і , то міняємо місцями символи та , .

6.2) Якщо , то ,якщо , то , інакше не змінюємо , .

6.3) Якщо , то видаляємо з , .

6.4) Якщо , то вставляємо після у , .

7) Так як ми записували зміни , то вже маємо шукану послідовність дій.

2. Необмежена відстань Дамерау-Левенштейна між рядками та , і оптимальна послідовність дій для перетворення у .

1) Для кожного символу алфавіту будемо зберігати останнє його входження у (якщо нумерувати з 1). 0 – якщо немає. Ініціалізація .

2) У матриці будемо зберігати значення k і f для кожної ітерації по .

3) Ініціалізуємо .

4) Ініціалізуємо .

5) Цикл . У будемо зберігати останнє входження у (якщо нумерувати з 1). 0 – якщо немає.

5.1) Ініціалізація .

5.2) Цикл . . . Обчислюємо .

5.2.1)

.

5.2.2) Якщо , то .

5.2.3) Якщо , то оновлюємо .

5.3) Оновлюємо .

6) – шукана відстань.

7) Нехай . Будемо її перетворювати та записувати ці перетворення.

8) Починаючи з (нижній правий кут матриці ), до поки та одночасно не будуть дорівнювати 0 (верхній лівий кут матриці ), обчислюємо

. (Не враховуємо компоненту, якщо індекс < 0, хоча б один з компонентів буде визначеним). Далі

8.1) . Якщо і , то видаляємо символи , транспонуємо та вставляємо у після , .

8.2) Якщо , то ,якщо , то , інакше не змінюємо , .

8.3) Якщо , то видаляємо з , .

8.4) Якщо , то вставляємо після у , .

9) Так як ми записували зміни , то вже маємо шукану послідовність дій.

**Складність**

Розмір вхідних даних – довжина рядків , та – та відповідно.

Обмежена відстань Дамерау-Левенштейна і оптимальна послідовність дій для перетворення: час – , пам’ять – .

Необмежена відстань Дамерау-Левенштейна і оптимальна послідовність дій для перетворення: час – , пам’ять – . Де - розмір

Маємо обчислення матриці у двох вкладених циклах.

**Мова програмування**

С++

**Модулі програми**

* long long osa\_dist(const std::string& s1, const std::string& s2, std::vector<std::string>& edit\_sequence,

std::vector<std::string>& edited\_strings)

Функція, яка повертає обмежену відстань Дамерау-Левенштейна (optimal string alignment distance), а у edit\_sequence записує послідовність операцій, у edited\_strings – строки після кожного перетворення.

* long long damerau\_levenshtein\_dist (const std::string& s1, const std::string& s2, std::vector<std::string>& edit\_sequence,

std::vector<std::string>& edited\_strings)

Функція, яка повертає необмежену відстань Дамерау-Левенштейна, а у edit\_sequence записує послідовність операцій, у edited\_strings – строки після кожного перетворення. Працює тільки для літер a-z.

**Інтерфейс користувача**

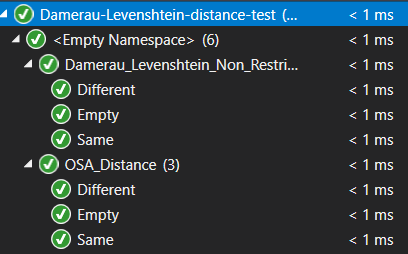
Вхідні дані вводяться з текстового файла і виводяться також в текстовий файл.

**Тестові приклади**

|  |  |
| --- | --- |
| **damerau-lavenshtein-dist\_input** | **damerau-lavenshtein-dist\_input** |
| 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 |

**Юніт-тести**

Програма проходить юніт-тести.

****

**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**

* https://en.wikipedia.org/wiki/Universal\_hashing
* <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://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>