Київський національний університет імені Тараса Шевченка

Факультет комп`ютерних наук та кібернетики

Кафедра інтелектуальних програмних систем

Алгоритми та складність

Завдання

Обчислити відстань Левенштейна-Дамерау, вивести на екран послідовність дій для перетворення першого рядка в другий

Гончаренко Ілля Сергійович

Київ - 2023

Завдання:

Обчислити відстань Левенштейна-Дамерау, вивести на екран послідовність дій для перетворення першого рядка в другий

Предметна область: 1)Редагування тексту, 2)розпізнавання мови, 3)керування базами даних.

Примітки: 1)Відстань Левенштейна-Дамерау широко використовується в редакторах тексту та системах автокорекції для визначення, наскільки схожі слова чи фрази та які саме операції потрібно виконати для перетворення одного тексту в інший.

2)В обробці природної мови відстань Левенштейна-Дамерау використовується для порівняння рядків та визначення схожості мовних конструкцій.

3)При роботі з базами даних, де зберігається текстова інформація, відстань Левенштейна-Дамерау може використовуватися для пошуку схожих записів чи виявлення дублікатів.

Теорія

Відстань Левенштейна-Дамерау — це міра відмінності між двома рядками символів. Вона визначається як мінімальна кількість елементарних операцій, таких як вставка, вилучення, заміна та транспозиція, необхідних для перетворення одного рядка в інший.

Матриця в стилі vector<vector<T>> є вектором векторів у мові програмування C++. Вона є способом представлення двовимірного масиву, де кожен вектор представляє один рядок матриці, а кожен елемент цього вектора - елемент цього рядка.

Транспозиція у математичному та програмувальному контексті означає обмін рядків і стовпців у матриці. Якщо у вас є матриця з розмірами m×n, то транспонована матриця матиме розміри n×m, і елементи, які були у рядках оригінальної матриці, тепер будуть у відповідних стовпцях та навпаки.

Для виведення послідовності дій можна використовувати зворотній прохід по матриці відстані та враховувати, які операції призводять до поточного значення відстані.

Це алгоритмічне завдання важливе для різних областей, включаючи обробку тексту, роботу з базами даних та автоматичне завершення тексту. Точний алгоритм може бути реалізований на різних мовах програмування за допомогою динамічного програмування або інших методів.

Алгоритми

1. Ініціалізація матриці D розміром (m+1) x (n+1), де m - довжина першого рядка, n - довжина другого.
2. Заповнення першого рядка та першого стовпчика числами від 0 до m та від 0 до n відповідно.
3. Перебір кожного символу з першого рядка і кожного символу з другого рядка:
    - Якщо символи співпадають, то D[i][j] = D[i-1][j-1] (береться значення по діагоналі).
    - Якщо символи відрізняються, D[i][j] = min(D[i-1][j] + 1, D[i][j-1] + 1, D[i-1][j-1] + 1) (береться мінімальне значення зліва, зверху і по діагоналі).

- Якщо враховується можливість транспозиції, перевіряється умова: якщо i, j > 1 та рівність символів у попередньому рядку і попередньому стовпчику, то D[i][j] = min(D[i][j], D[i-2][j-2] + 1).

4. Значення в правому нижньому куті матриці D[m][n] буде відстанню Левенштейна-Дамерау.

Для відображення послідовності дій можна використати матрицю, зберігаючи інформацію про оптимальні операції для кожної комірки. Наприклад, для відстеження оптимального шляху можна використовувати додаткову матрицю, що зберігає інформацію про операції: вставка, видалення, заміна, транспозиція або збіг символів.

Це складний алгоритм, тому його реалізація в коді може зайняти багато часу і ресурсів, але він надзвичайно корисний для різних задач, таких як автозавершення в текстових редакторах або пошук похожих слів.

Цей код виконує вище описані дії:

```
function levenshteinDamerauDistance(s1, s2):
    m = length(s1)
    n = length(s2)
    // Ініціалізація матриці D
    D = createMatrix(m+1, n+1)
    // Ініціалізація першого рядка та першого стовпчика
    for i from 0 to m:
        D[i][0] = i
    for j from 0 to n:
        D[0][j] = j
```

```
    // Заповнення матриці D
    for i from 1 to m:
        for j from 1 to n:
            cost = 1
            if s1[i-1] == s2[j-1]:
                cost = 0
            // Обчислення мінімального відстані
            D[i][j] = min(
                D[i-1][j] + 1,     // видалення
                D[i][j-1] + 1,     // вставка
                D[i-1][j-1] + cost // заміна
            )
            // Врахування транспозиції
            if i > 1 and j > 1 and s1[i-1] == s2[j-2] and s1[i-2] == s2[j-1]:
                D[i][j] = min(D[i][j], D[i-2][j-2] + cost)
    // Виведення відстані Левенштейна-Дамерау
    print("Відстань Левенштейна-Дамерау:", D[m][n])
    // Виведення послідовності дій
    actions = backtraceActions(D, s1, s2)
    print("Послідовність дій:", actions)
function backtraceActions(D, s1, s2):
```

```
actions = []
i = length(s1)
j = length(s2)
while i > 0 or j > 0:
    if i > 0 and j > 0 and s1[i-1] == s2[j-1]:
        // збіг символів, нічого не робимо
        i = i - 1
        j = j - 1
    else:
        cost = 1
        if s1[i-1] == s2[j-1]:
            cost = 0
        // Перевірка оптимальної операції
        if i > 0 and D[i][j] == D[i-1][j] + 1:
            actions.add("Видалити '" + s1[i-1] + "' з s1")
            i = i - 1
        else if j > 0 and D[i][j] == D[i][j-1] + 1:
            actions.add("Вставити '" + s2[j-1] + "' в s1")
            j = j - 1
        else:
            actions.add("Замінити '" + s1[i-1] + "' на '" + s2[j-1] + "'")
```

```
            i = i - 1

            j = j - 1

    return reverse(actions)
```

Складність алгоритму

Складність алгоритму Левенштейна-Дамерау в середньому є $O(m \cdot n)$, де m та n - довжини порівнюваних рядків.

Оскільки потрібно заповнити матрицю розміром $(m+1)\times(n+1)$ та для кожної комірки виконувати деякі обчислення (враховуючи умови для вставки, видалення, заміни символів та, якщо враховується транспозиція, додаткова перевірка), складність стає квадратичною.

Мова реалізації алгоритму C++

Модулі програми:

- **int LevenshteinDistance(string str1, string str2)**

Ця функція реалізує алгоритм Левенштейна-Дамерау для обчислення відстані між двома рядками str1 та str2. Окрім того, вона виводить на екран послідовність операцій, які необхідно виконати для перетворення рядка str1 в рядок str2.

Основні етапи алгоритму:

1. **Ініціалізація матриці dp:** Створюється матриця dp розміром (len1 + 1) x (len2 + 1), де len1 та len2 - довжини рядків str1 та str2 відповідно. Матриця dp використовується для зберігання відстаней між підстрічками.

2. **Заповнення матриці dp:** Вкладена подвійна ітерація проймає всі комірки матриці dp і визначає відстань між підстрічками str1[0...i-1] та str2[0...j-1]. Використовуються операції вставки, видалення, заміни та, якщо враховується транспозиція (Damerau), додаткова перевірка для операції транспозиції.
3. **Виведення послідовності операцій:** Після заповнення матриці dp, функція проходиться по ній знизу вгору, визначаючи послідовність операцій, які необхідно виконати для перетворення str1 в str2. Операції включають вставку, видалення, заміну та транспозицію. Результат (відстань Левенштейна-Дамерау) повертається як значення функції.
4. **Виведення на екран інформації про операції:** Кожна операція виводиться на екран разом з відомостями про те, які саме символи або позиції змінюються.

Цей алгоритм корисний для визначення "відстані" між двома рядками та виведення операцій, які необхідно виконати для перетворення одного рядка в інший.

Інтерфейс користувача

Цей код має консольний інтерфейс користувача. Він використовує введення та виведення через консоль для введення рядків та виведення результатів.

Основні етапи роботи програми:

Обчислення відстані: Викликається функція LevenshteinDistance, яка обчислює відстань Левенштейна-Дамерау між двома рядками str1 та str2. Функція також виводить на екран послідовність операцій, які необхідно виконати для перетворення str1 в str2.

Виведення результатів: Виводиться відстань Левенштейна-Дамерау між рядками str1 та str2. Також виводяться операції, які необхідно виконати для перетворення рядка str1 в рядок str2.

Цей інтерфейс дозволяє користувачу вводити свої рядки та перевіряти, наскільки вони схожі за відстанню Левенштейна-Дамерау. При цьому виводиться також інформація про операції, які потрібно виконати для перетворення одного рядка в інший.

Тестові приклади

1. **Приклад на однакові рядки:**

   Вхідні дані:

```
string str1 = "hello";
string str2 = "hello";
```

   Вихідні дані:

```
Відстань Левенштейна-Дамерау між рядками 'hello' та 'hello': 0
```

2. **Приклад на рядки з однаковими символами, але в різному порядку:**

   Вхідні дані:

```
string str1 = "abcd";
string str2 = "dcba";
```

   Вихідні дані:

```
Replace 'd' at position 4 with 'a'
Transpose 'b' at position 2 and 'c' at position 3
Replace 'a' at position 1 with 'd'
Відстань Левенштейна-Дамерау між рядками 'abcd' та 'dcba': 3
```

3. **Приклад на додавання та вилучення символів:**

   Взідні дані:

```
string str1 = "kitten";
string str2 = "sitting";
```

   Вихідні дані:

```
Insert 'g' at position 7
Replace 'e' at position 5 with 'i'
Replace 'k' at position 1 with 's'
Відстань Левенштейна-Дамерау між рядками 'kitten' та 'sitting': 3
```

4. **Прилад на заміну символів та транспозицію:**

   Вхідні дані:

```
string str1 = "hello";
string str2 = "world";
```

Вихідні дані:

```
Replace 'o' at position 5 with 'd'
Replace 'l' at position 3 with 'r'
Replace 'e' at position 2 with 'o'
Replace 'h' at position 1 with 'w'
Відстань Левенштейна-Дамерау між рядками 'hello' та 'world': 4
```

Ці приклади допоможуть вам перевірити, чи працює функція коректно та чи правильно вона визначає відстань та послідовність операцій для різних пар рядків.

Висновки

Отже, у даній роботі, було описано алгоритм обчислення відстані Левенштейна-Дамерау, та виведення на екран послідовність дій для перетворення першого рядка в другий.

Цей код реалізує алгоритм Левенштейна-Дамерау для обчислення відстані між двома рядками. Використовується матриця dp для зберігання відстаней між підстрічками та визначення оптимальних операцій. Виводиться послідовність операцій, які необхідно виконати для перетворення одного рядка в інший. Код може бути використаний для порівняння будь-яких рядків та визначення відстані між ними.

Можливе використання алгоритму для автозавершення тексту, пошуку схожих слів, тощо. Виведення послідовності операцій дозволяє зрозуміти, які конкретно зміни виконані для перетворення рядків. Це може бути корисним для розуміння та відстеження змін в текстових даних.

Використані літературні джерела

- https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance#Algorithm
- https://habr.com/ru/articles/676858/
- https://www.youtube.com/watch?v=4TgAdLQ9oVY