Содержание

Введение 3
1. Описание алгоритмов 4
1.1 Нерекурсивный алгоритм для расстояния Левенштейна 4
1.2 Нерекурсивный алгоритм для расстояния Дамерау-Левенштейна 4
1.3 Рекурсивный алгоритм для расстояния Дамерау-Левенштейна 5
1.4 Рекурсивный алгоритм с кэшем для расстояния Дамерау-Левенштейна 5
2. Средства реализации и архитектура 6
3. Сложности реализованных алгоритмов 6
3.1 Временная сложность 6
3.2 Затрачиваемая память 7
Заключение 9
Список используемых источников 9

Введение

Редакционное расстояние – это минимальное количество операций, необходимых для преобразования одной строки в другую. Оно играет ключевую роль в таких приложениях, как исправление орфографических ошибок, сравнение геномных последовательностей, машинный перевод и анализ текстов. Наиболее известными подходами к решению этой задачи являются алгоритмы Левенштейна и Дамерау-Левенштейна, которые учитывают различные наборы редакционных операций.

Метод Левенштейна оперирует тремя базовыми операциями: вставкой, удалением и заменой символов. Его расширенная версия — метод Дамерау-Левенштейна — дополнительно включает транспозицию соседних символов, что позволяет учесть новый тип ошибки, где опечатки часто связаны с перестановкой букв.

Цель данной работы — реализация и сравнительный анализ алгоритмов Левенштейна и Дамерау-Левенштейна в различных вариантах: итерационном, рекурсивном и рекурсивном с кэшированием. В рамках исследования оцениваются временная сложность и использование памяти для каждого алгоритма, что позволяет определить оптимальные условия их применения.

Задачи:

1) формально описать алгоритмы Левенштейна и Дамерау–Левенштейна;
2) реализовать итеративный алгоритм нахождения расстояния Левенштейна;
3) реализовать рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна;
4) реализовать рекурсивную версию с кэшированием;
5) провести сравнительный анализ – оценить временные затраты и использование памяти для всех реализаций.

1. Описание алгоритмов

1.1 Нерекурсивный алгоритм для расстояния Левенштейна

Расстояние Левенштейна (редакционное расстояние) между двумя строками в теории информации и компьютерной лингвистике — это минимальное количество операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую [1].

Алгоритм расстояния Левенштейна оперирует тремя основными редакционными операциями, с помощью которых одна строка преобразуется в другую:

1. Insert (вставка) — вставка одного символа в строку;

2. Delete (удаление) — удаление одного символа из строки;

3. Replace (замена) — замена одного символа другим.

У каждой этой операции штраф равен 1. Также вводится обозначение Match (совпадение) – со штрафом 0 за отстутствие операции.

Рассматривается матрица D размером (m+1) × (n+1), где элемент D[i][j] соответствует минимальному количеству операций, необходимых для преобразования первых i символов строки s1 в первые j символов строки s2. Инициализация матрицы:

D[0][j] = j для всех j от 0 до n,

D[i][0] = i для всех i от 0 до m.

Далее, для всех i от 1 до m и j от 1 до n, элементы матрицы рассчитываются по следующему рекуррентному соотношению:

$$\begin{gathered} \mathrm{D[i][j]} = \min(\\ \mathrm{D[i-1][j]} + 1, (\mathrm{D}) \\ \mathrm{D[i][j-1]} + 1, (\mathrm{I}) \\ \mathrm{D[i-1][j-1]} + \mathrm{cost})(\mathrm{R}) \end{gathered}$$

где cost = 0, если s1[i-1] = s2[j-1], иначе cost = 1.

Временная сложность: O(m × n), что связано с необходимостью заполнения всех ячеек матрицы, каждая из которых требует постоянное время на вычисление минимума из трёх значений [1].

1.2 Нерекурсивный алгоритм для расстояния Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна между двумя строками, состоящими из конечного числа символов — это минимальное число операций вставки,

удаления, замены одного символа и транспозиции двух соседних символов, необходимых для перевода одной строки в другую.

Алгоритм использует матрицу D размером (m+1) × (n+1), где D[i][j] вычисляется по формуле:

$$\begin{gathered} D[i][j] = \min(\\ D[i-1][j] + 1, (D) \\ D[i][j-1] + 1, (I) \\ D[i-1][j-1] + \text{cost}, (R) \\ D[i-2][j-2] + 1 \text{ if } i > 1, j > 1: \\ s1[i-1] == s2[j-2], s1[i-2] == s2[j-1]) \ (T) \end{gathered}$$

где cost = 0, если s1[i-1] = s2[j-1], иначе 1.

Временная сложность: O(m × n) [1].

1.3 Рекурсивный алгоритм для расстояния Дамерау-Левенштейна

Суть рекурсивного алгоритма заключается в том, что каждый раз при обработке основных строк s1[1..i] и s2[1..j] мы вызываем алгоритм для обработки трех вариантов изменения строк: D(s1[1..i - 1], s2[1..j]), D(s1[1..i - 1], s2[1..j - 1]), D(s1[1..i], s2[1..j - 1]). Операции соответствуют удалению, замене и вставке символа соответственно, стоимость которых равна 1. Кроме того, если текущий и предыдущий символы в s1 совпадают с предыдущим и текущим символами в s2, алгоритм проверяет возможность транспозиции (перестановки соседних символов) также со стоимостью 1.

Однако у данного варианта есть значительный недостаток — повторные вычисления. Каждый вызов будет обрабатываться индивидуально, а значит для одинаковых входных параметров пересчет будет вестись снова и снова. Чтобы избавиться от многократного определения расстояния можно использовать кэш. [1]

1.4 Рекурсивный алгоритм с кэшем для расстояния Дамерау-Левенштейна

В основе алгоритма лежит использование матрицы кэша, где каждый элемент D[i][j] хранит минимальное количество операций для преобразования первых i символов строки s1 в первые j символов строки s2. Изначально все элементы матрицы инициализируются значением «бесконечность». При первом обращении к D[i][j] алгоритм проверяет, сохранено ли уже значение в кэше. Если элемент равен бесконечности — он еще не рассчитан, и алгоритм приступает к рекурсивному вычислению. В противном случае возвращается готовое значение из кэша, что исключает повторные вычисления [1].

Рекурсивная логика учитывает четыре возможные операции:

1. Удаление (D) — преобразование s1[1..i-1] в s2[1..j] с добавлением штрафа 1;
2. Вставка (I) — преобразование s1[1..i] в s2[1..j-1] с добавлением штрафа 1;
3. Замена или совпадение (R/M) — если последние символы s1[i] и s2[j] совпадают (M), стоимость замены равна 0, иначе (R) — 1;
4. Транспозиция (T) — если i > 1 и j > 1, а также s1[i] = s2[j-1] и s1[i-1] = s2[j], то рассматривается вариант перестановки соседних символов с добавлением штрафа 1.

Для каждой подзадачи алгоритм выбирает минимальную стоимость из всех возможных операций, сохраняет результат в кэш и возвращает его. Например, при транспозиции двух символов в строках «ст» и «тс», алгоритм учтет это как одну операцию вместо двух замен, сокращая общее расстояние.

Оптимизация через кэширование снижает вычислительную сложность с экспоненциальной (как в простой рекурсии) до O(m·n), где m и n — длины строк. Это достигается за счет того, что каждая уникальная пара (i, j) вычисляется только один раз, а последующие обращения к ней используют сохраненное значение. Таким образом, алгоритм сочетает логическую прозрачность рекурсивного подхода с эффективностью динамического программирования.

2. Средства реализации и архитектура

Технические характеристики устройства, на котором выполнялись замеры:

1) операционная система Windows 11;
2) оперативная память 32 ГБ;
3) процессор Intel(R) Core(TM) i7-7600U @ 2.80 ГГц , архитектура x64;
4) компилятор MinGW;
5) среда разработки Clion, язык программирования: C++;
6) Ноутбук подключен к сети электропитания;

Условия замеров процессорного времени следующие:

1) система нагружена только средой разработки CLion и тестируемой программой;
2) для замера времени выполнения реализации алгоритмов использовался счетчик тактов процессора (TSC) через функцию __rdtsc() [2].

3. Сложности реализованных алгоритмов

3.1 Временная сложность

В таблице 1 демонстрируются результаты замеров затрат времени для реализации описанных ранее алгоритмов.

Таблица 1. Результаты замеров времени работы реализованных алгоритмов в тактах процессора

Длина	Левенштейн (такты)	Дамерау-Левенштейн итер (такты)	Дамерау - Левенштейн рекурс (такты)	Дамерау-Левенштейн с кэш (такты)
1	3213.04	3104.63	170.99	348.31
2	4164.99	4056.08	926.89	1115.62
3	5458.74	5993.29	4518.52	2479.68
4	9497.41	10571.5	29281.17	6643.28
5	13413.2	13805.6	167308.29	9088.91

Рекурсивная реализация алгоритма Дамерау-Левенштейна без использования кэша демонстрирует критическую неэффективность. Уже для строк длиной 5 символов время выполнения достигает 167308 тактов — это в 12 раз медленнее, чем у итеративной версии того же алгоритма (13805 тактов). Причина кроется в экспоненциальном росте вычислительной сложности: каждый рекурсивный вызов порождает множество дублирующихся подзадач. На практике это делает алгоритм неработоспособным для строк длиннее 5 символов — при тестировании на длине больше 5 символов скрипт не завершал работу в разумные сроки. Наименее же затратная по времени это реализация алгоритма Дамерау-Левенштейна с кэшем (в 18 раз быстрее базовой рекурсивной версии). Итеративные реализации показывают сопоставимую в ±15% производительность.

3.2 Затрачиваемая память

Во время эксперимента была произведена оценка объёма памяти, необходимого для работы различных реализаций алгоритмов вычисления редакционного расстояния в зависимости от длины входных строк. Результаты сведены в таблицу, где значения представлены в килобайтах.

Таблица 2. Затраты памяти для реализаций алгоритмов в зависимости от длины строк в килобайтах

Длина	Левенштейн (Кб)	Дамерау-Левенштейн итер (Кб)	Дамерау - Левенштейн рекурс (Кб)	Дамерау-Левенштейн с кэш (Кб)
1	16	16	36	48
2	36	36	36	108
3	64	64	36	192
4	100	100	36	300

5	144	144	36	432
10	484	484	36	1452

Как видно из таблицы, итеративные алгоритмы (как Левенштейна, так и Дамерау–Левенштейна) демонстрируют квадратичную зависимость потребления памяти от длины строки. Это объясняется использованием двумерной матрицы размером (n+1)×(n+1)(n+1)×(n+1), в которой хранятся промежуточные значения. Каждый элемент матрицы занимает 4 байта, так как используется тип int. Таким образом, с увеличением длины строки рост объёма используемой памяти становится заметным.

Рекурсивная реализация алгоритма Дамерау–Левенштейна без кэширования, напротив, показывает стабильное потребление памяти — около 36 килобайт во всех тестируемых случаях. Это связано с тем, что алгоритм не использует дополнительных структур данных и расходует память только на стек вызовов. Однако следует отметить, что в силу экспоненциальной сложности, данный вариант применим только для коротких строк.

Наиболее ресурсоёмким оказался рекурсивный алгоритм с кэшированием. Помимо использования стека вызовов, он дополнительно хранит матрицу промежуточных результатов, аналогичную по размеру матрице итеративных алгоритмов. Это приводит к наибольшему объёму затрачиваемой памяти, который также увеличивается квадратично при росте длины входных строк.

Таким образом, в контексте оценки потребления памяти, наименее затратным является рекурсивный подход без кэширования (при малой длине строк), тогда как наиболее затратным — рекурсивный алгоритм с кэшированием. Итеративные алгоритмы занимают промежуточное положение и демонстрируют предсказуемый рост использования ресурсов.

Заключение

Цель лабораторной работы – реализация и исследование алгоритмов вычисления редакционного расстояния Левенштейна и Дамерау–Левенштейна – была успешно достигнута. Все поставленные задачи выполнены в полном объёме, что позволило провести сравнительный анализ и сделать следующие выводы:

1) итеративный метод Левенштейна продемонстрировал затрату O(m × n) по времени и памяти;
2) рекурсивный метод Дамерау-Левенштейна оказался непригодным для строк длиннее 5 символов из-за экспоненциального роста времени, затраченного на расчёт расстояния;
3) рекурсивный алгоритм с кэшированием, в котором оптимизация через кэш снизила сложность до O(m × n), сделав его конкурентоспособным с итеративным аналогом;
4) итеративные методы и рекурсия с кэшем показали близкие результаты (разница в ±15%), тогда как базовая рекурсия оказалась в 18 раз медленнее для n = 5.
5) итеративные подходы и рекурсия с кэшем требуют O(m × n) памяти, что ограничивает их для длинных строк. Классическая рекурсия, несмотря на фиксированные 36 Кб, неприменима на практике из-за времени выполнения.

Список используемых источников

1. Задача о редакционном расстоянии // Викиконспекты ИТМО [Электронный ресурс] URL: https://neerc.ifmo.ru/wiki/index.php?title=Задача о редакционном расстоянии, алгоритм Вагнера-Фишера (дата обращения: 24.04.2025).
2. *Microsoft. RDTSC (Read Time-Stamp Counter)* // Microsoft Learn [Электронный ресурс] URL: https://learn.microsoft.com/ru-ru/cpp/intrinsics/rdtsc?view=msvc-170 (дата обращения: 24.04.2025).