Лабораторная работа №1
По курсу: "Анализ алгоритмов"
По теме: "Расстояние Левенштейна"

Студент: Кондрашова О.П.
Группа: ИУ7-55Б
Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Содержание

Введение

Цель работы: изучение метода динамического программирования на материале алгоритмов нахождения расстояния Левенштейна и расстояния Дамерау-Левенштейна.

В данной работе рассматриваются три алгоритма: итеративный алгоритм нахождения расстояния Левенштейна, расстояния Дамерау-Левенштейна и рекурсивная реализация алгоритма нахождения расстояния Дамерау-Левенштейна.

Задачи данной работы:

1. изучение алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна, нахождения расстояния между строками;
2. применение метода динамического программирования для матричной реализации указанных алгоритмов;
3. получение практических навыков реализации указанных алгоритмов: двух алгоритмов в матричной версии и одного из алгоритмов в рекурсивной версии;
4. сравнительный анализ линейной и рекурсивной реализаций выбранного алгоритма определения расстояния между строками по затрачиваемым ресурсам (времени и памяти);
5. экспериментальное подтверждение различий во временно́й эффективности рекурсивной и нерекурсивной реализаций выбранного алгоритма определения расстояния между строками при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующихся длинах строк;
6. описание и обоснование полученных результатов в отчете о выполненной лабораторной работе, выполненного как расчётно-пояснительная записка к работе.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы алгоритмов.

1.1 Алгоритм Левенштейна

Расстояние Левенштейна (также редакционное расстояние или дистанция редактирования) 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.

Область применения данных алгоритмов:

1. выявление и исправление ошибок в словах при вводе, в поисковиках, в различных базах данных, в программах автоматического определения текста и т. п.
2. сравнение текстовых файлов утилитами diff и подобных ей;
3. сравнение генов, белков и хромосом в биоинформатике.

Данный алгоритм использует понятие «цена операции». У каждой из операций вставки, замены или удаления «цена» равняется единице. У совпадения символа - нулю. Нахождение расстояния Левенштейна сводится к нахождению последовательности операций, приводящих одну строку к другой, с минимальной суммарной ценой.

Классификация разрешенных операций и штрафы на выполнение операции.

1. замена символа - R = 1;
2. вставка символа - I = 1;
3. удаление символа - D = 1;
4. совпадение символа - M = 0.

Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда расстояние Левенштейна можно подсчитать по следующей рекуррентной формуле:

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & j = 0, i > 0 \\ j, & i = 0, j > 0 \\ min \begin{cases} D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + (S1[i] \neq S2[j]) \end{cases} & j > 0, i > 0 \end{cases}$$

1.2 Алгоритм Дамерау-Левенштейна

Расстояние Дамерау–Левенштейна между двумя словами — минимальное количество элементарных операций, которые необходимо совершить, чтобы получить одно слово из другого. Алгоритм является модификацией алгоритма расстояния Левенштейна. К имеющимся трем элементарным операциям добавляется операция перестановки двух соседних символов (транспозиция).

Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда расстояние Левенштейна $d(S_1, S_2)$ можно подсчитать по следующей рекуррентной формуле $d(S_1, S_2) = D(M, N)$, где:

$$D(i,j) = \begin{cases} 0, & i = 0, j = 0 \\ i, & i > 0, j = 0 \\ j, & i = 0, j > 0 \\ min \begin{cases} D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + m(S_1[i], S_2[i]), \\ D(i-2, j-2) + m(S_1[i], S_2[i]), \end{cases} & \begin{array}{l} \text{, если } i, j > 0 \\ \text{и } S_1[i] = S_2[j-1] \\ \text{и } S_1[i-1] = S_2[j] \end{array} \\ min \begin{cases} D(i, j-1) + 1, \\ D(i-1, j) + 1, \\ D(i-1, j-1) + m(S_1[i], S_2[i]), \end{cases} & \text{, иначе} \end{cases}$$

1.3 Вывод

В аналитической части были рассмотрены алгоритмы нахождения расстояния Левенштейна и его усовершенствованный алгоритм нахождения расстояния Дамерау-Левенштейна, принципиальная разница которого — наличие транспозиции. Также были рассмотрены области применения данных алгоритмов.

2 Конструкторская часть

В данной части будут рассмотрены схемы алгоритмов.

2.1 Схемы алгоритмов

Рассмотрим итеративные алгоритмы нахождения расстояния Левенштейна, Дамерау-Левенштейна и рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна.

Схема алгоритма нахождения расстояния Левенштейна.

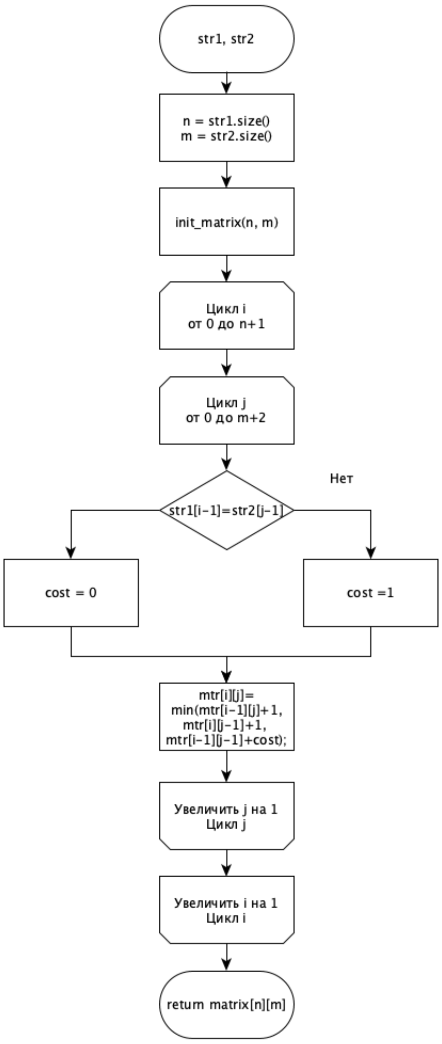

Рис. 1: Матричный алгоритм нахождения расстояния Левенштейна

Схема алгоритма нахождения расстояния Дамерау-Левенштейна.

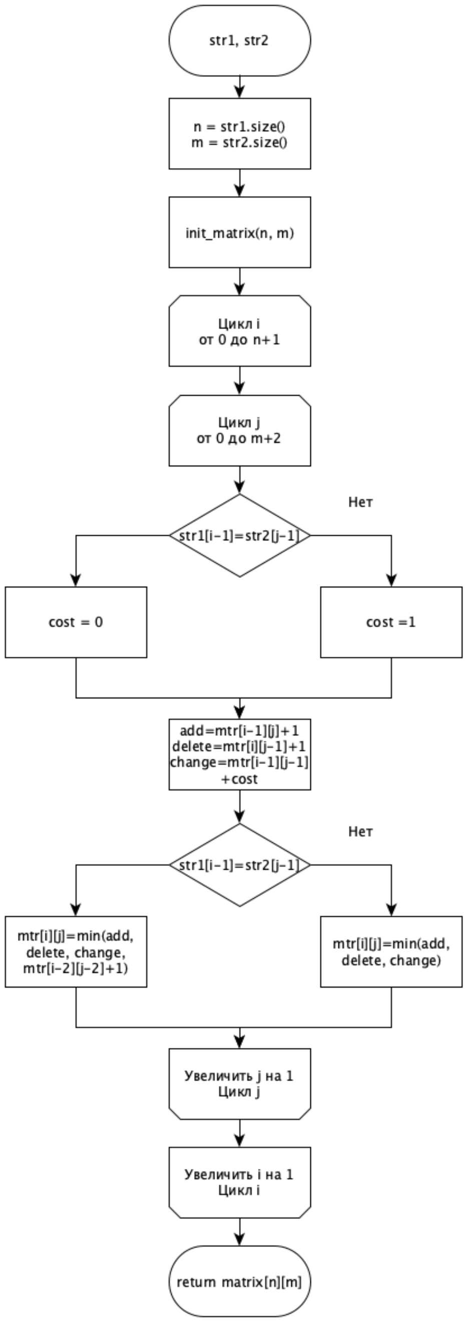

Рис. 2: Матричный алгоритм нахождения расстояния Дамерау-Левенштейна

Схема рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна.

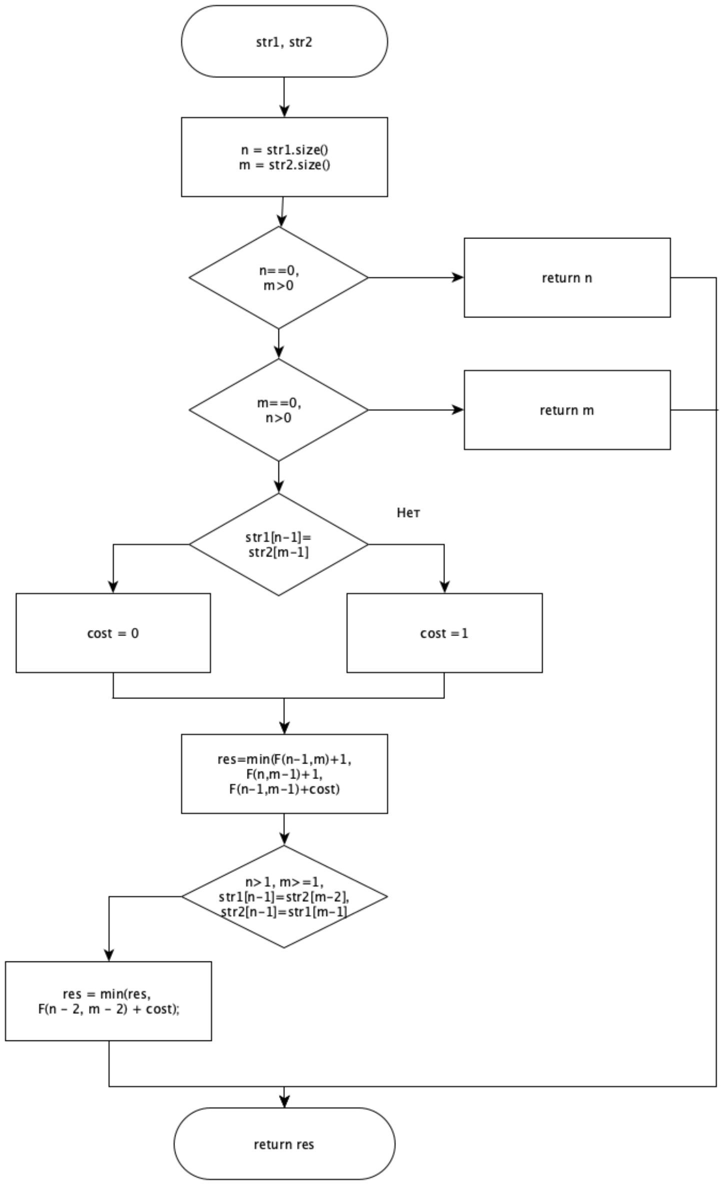

Рис. 3: Рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна

2.2 Вывод

В данном разделе были рассмотрены схемы алгоритмов нахождения расстояния Левенштейна, Дамерау-Левенштейна, а также рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна.

3 Технологическая часть

В этом разделе будут изложены требования к программному обеспечению, средства реализации, а также представлен листинг кода.

3.1 Требования к программному обеспечению

Входные данные: str1 - первое слово, str2 - второе слово.
Выходные данные: значение расстояния между двумя словами.

Рис. 4: IDEF0 диаграмма нахождения расстояния Левенштейна

3.2 Средства реализации

Данная программа разработана на языке C++, поддерживаемом многими операционными системами. Проект выполнен в среде Xcode.

Для замера процессорного времени используется функция, возвращающая количество тиков.

Листинг 1: Функция замера количества тиков

```
unsigned long long tick()
{
    unsigned long long d;
    __asm__ __volatile__ ("rdtsc" : "=A" (d));
    return d;
}
```

3.3 Листинги функций

Листинг 2: Алгоритм нахождения расстояния Левенштейна

```
int Levenshtein::levenshtein()
{
    matrix.clear();
    int n = str1.size();
    int m = str2.size();
    init_matrix(n, m);

    int cost;
    for (int i = 1; i < n + 1; i++)
    {
        for (int j = 1; j < m + 1; j++)
        {
            cost = str1[i - 1] == str2[j - 1] ? 0 : 1;
            matrix[i][j] = min(min(matrix[i - 1][j] + 1,
                matrix[i][j - 1] + 1), matrix[i - 1][j - 1] +
                cost);
        }
    }
    return matrix[n][m];
}
```

Листинг 3: Алгоритм нахождения расстояния Дамерау-Левенштейна

```
int Levenshtein::damerau_levenshtein()
{
    matrix.clear();
    int n = str1.size();
    int m = str2.size();
    init_matrix(n, m);

    int cost, transp;
    for (int i = 1; i < n + 1; i++)
    {
        for (int j = 1; j < m + 1; j++)
        {
            cost = str1[i - 1] == str2[j - 1] ? 0 : 1;
            if (i > 1 && j > 1 && str1[i - 1] == str2[j - 2]
                && str1[i - 2] == str2[j - 1])
            {
                transp = matrix[i - 2][j - 2] + 1;
                matrix[i][j] = min(min(matrix[i - 1][j] + 1,
                    matrix[i][j - 1] + 1), min(matrix[i - 1][j
                    - 1] + cost, transp));
            }

            else
            {
                matrix[i][j] = min(min(matrix[i - 1][j] + 1,
                    matrix[i][j - 1] + 1), matrix[i - 1][j -
                    1] + cost);
            }
        }
    }
    return matrix[n][m];
}
```

Листинг 4: Рекурсивный Алгоритм нахождения расстояния Дамерау-Левенштейна

```
int Levenshtein::damerau_levenshtein_recursive(int i, int j)
{
    if (!i)
        return j;
    if (!j)
        return i;

    int res, cost;

    cost = str1[i - 1] == str2[j - 1] ? 0 : 1;

    res = min(min(damerau_levenshtein_recursive(i - 1, j) + 1,
         damerau_levenshtein_recursive(i, j - 1) + 1),
        damerau_levenshtein_recursive(i - 1, j - 1) + cost);

    if (i > 1 and j >= 1 and str1[i - 1] == str2[j - 2] and
        str2[i - 1] == str1[j - 1])
    {
        res = min(res, damerau_levenshtein_recursive(i - 2, j
            - 2) + cost);
    }

    return res;
}
```

3.4 Сравнительный анализ потребляемой памяти

В алгоритме нахождения расстояния Левенштейна используются:

Таблица 1: Память, используемая в алгоритме нахождения расстояния Левенштейна

Структура данных	Занимаемая память (байты)
Матрица (vector)	24
2 строки	48
3 вспомогательные переменные (int)	12
2 счетчика (int)	8

В алгоритме нахождения расстояния Дамерау-Левенштейна используются:

Таблица 2: Память, используемая в алгоритме нахождения расстояния Дамерау-Левенштейна

Структура данных	Занимаемая память (байты)
Матрица (vector)	24
2 строки	48
4 вспомогательные переменные (int)	16
2 счетчика (int)	8

В рекурсивном алгоритме нахождения расстояния Дамерау-Левенштейна используются:

Таблица 3: Память, используемая в рекурсивном алгоритме нахождения расстояния Дамерау-Левенштейна

Структура данных	Занимаемая память (байты)
2 строки	48
4 вспомогательные переменные (int)	16

Максимальная глубина рекурсивного вызова функции - сумма длин двух слов.

3.5 Оценка потребляемой памяти

Оценим алгоритмы на словах длинной 4 символа:

Таблица 4: Память, потребляемая структурами в алгоритме нахождения расстояния Левенштейна

Структура данных	Занимаемая память (байты)
Матрица (vector)	184
2 строки	48
3 вспомогательные переменные (int)	48
2 счетчика (int)	24
Всего:	304

Таблица 5: Память, потребляемая структурами в алгоритме нахождения расстояния Дамерау-Левенштейн

Структура данных	Занимаемая память (байты)
Матрица (vector)	184
2 строки	48
4 вспомогательные переменные (int)	64
2 счетчика (int)	24
Всего:	320

Таблица 6: Память, потребляемая структурами в рекурсивном алгоритме нахождения расстояния Дамерау-Левенштейна

Структура данных	Занимаемая память (байты)
2 строки	48
4 вспомогательные переменные (int)	16 * (4 + 4) (максимальная глубина вызовов функции) = 64
Всего:	176

Оценим алгоритмы на словах длинной 1000 символов:

Таблица 7: Память, потребляемая структурами в алгоритме нахождения расстояния Левенштейна

Структура данных	Занимаемая память (байты)
Матрица (vector)	24024024
2 строки	48000
3 вспомогательные переменные (int)	12000
2 счетчика (int)	8000
Всего:	24092024

Таблица 8: Память, потребляемая структурами в алгоритме нахождения расстояния Дамерау-Левенштейна

Структура данных	Занимаемая память (байты)
Матрица (vector)	24024024
2 строки	48000
4 вспомогательные переменные (int)	16000
2 счетчика (int)	8000
Всего:	24096024

Таблица 9: Память, потребляемая структурами в рекурсивном алгоритме нахождения расстояния Дамерау-Левенштейна

Структура данных	Занимаемая память (байты)
2 строки	48000
4 вспомогательные переменные (int)	16 * (1000 +1000)(максимальная глубина вызовов функции) =
Всего:	80000

3.6 Вывод

В данном разделе была представлена реализация алгоритмов нахождения расстояния Левенштейна, Дамерау-Левенштейна, а также рекурсивный алгоритм нахождения расстояния Дамерау-Левенштейна. Произведен анализ по потребляемой памяти каждым из алгоритмов в входе которого был сделан вывод о том, что рекурсивный алгоритм по мере увеличения длин строк начинает выигрывать по количеству занимаемой памяти.

4 Исследовательская часть

В данной части представлены результаты исследования быстродействия алгоритмов и затраченной памяти.

4.1 Тесты

Цифры в строках "Полученный результат"и "Ожидаемый результат"соответствуют расстоянию, найденному по алгоритму нахождения расстояния Левенштейна, Дамерау-Левенштейна и рекурсивной реализацией алгоритма нахождения расстояния Дамерау-Левенштейна соответственно.

Таблица 10: Таблица тестовых данных

№	Первое слово	Второе слово	Полученный результат	Ожидаемый результат
1			0 0 0	0 0 0
2	abc		3 3 3	3 3 3
3		abc	3 3 3	3 3 3
4	tup	dub	2 2 2	2 2 2
5	skat	kot	2 2 2	2 2 2
6	razvlechenie	uvlechenie	3 3 3	3 3 3
7	hate	hat	1 1 1	1 1 1
8	don	dun	1 1 1	1 1 1
9	mouse	mouse	0 0 0	0 0 0
10	qwer	asdf	4 4 4	4 4 4
11	qw	asdf	4 4 4	4 4 4
12	qwe	qaz	2 2 2	2 2 2
13	zaba	laba	1 1 1	1 1 1
14	ollo	lool	3 3 2	3 3 2
15	goda	pogoda	2 2 2	2 2 2

Все тесты пройдены успешно.

4.2 Постановка эксперимента по замеру времени

Для временного сравнительного анализа рекурсивной реализации алгоритма нахождения расстояния Левенштейна, алгоритма нахождения расстояния Дамерау-Левенштейна и рекурсивного алгоритма нахождения расстояния Дамерау-Левенштейна использовалась методика многочисленных запусков программы слов одинаковой длины (от 1 до 7 символов). Для расчета среднего времени работы алгоритмов будет использована формула $t = \frac{T_n}{N}$, где t – время выполнения, N – количество замеров. Было проведено по 20 замеров для каждой длины.

4.3 Сравнительный анализ на основе экспериментальных данных

Таблица 11: Время работы алгоритмов (в тиках)

Длина	Левенштейн	Дамерау-Левенштейн	Рек. Дамерау-Левенштейн
1	7102	7069	99
2	17362	15165	690
3	24143	23296	17972
4	35034	35462	15580
5	33133	34103	64178
6	46081	45632	416267
7	57429	52080	1888863

Сравнение итеративных алгоритмов нахождения расстояния Левенштейна и Дамерау-Левенштейна:

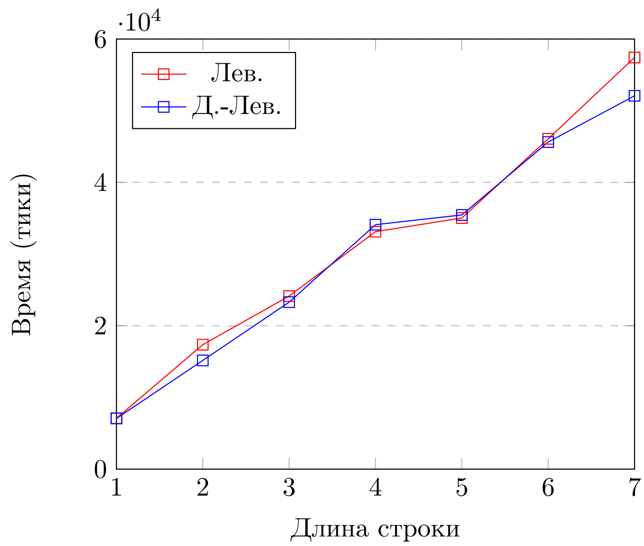

Рис. 5: Графики скорости работы итеративных алгоритмов

Сравнение итеративного и рекурсивного алгоритмов нахождения расстояния Дамерау-Левенштейна:

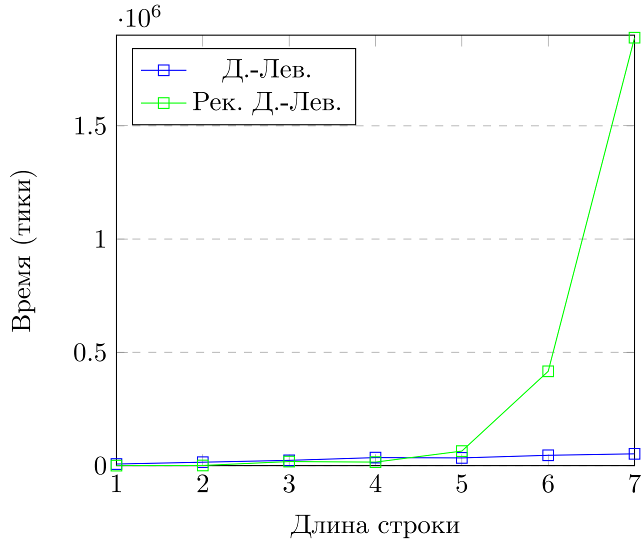

Рис. 6: Графики скорости работы итеративной и рекурсивной реализаций алгоритма нахождения расстояния Дамерау-Левенштейна

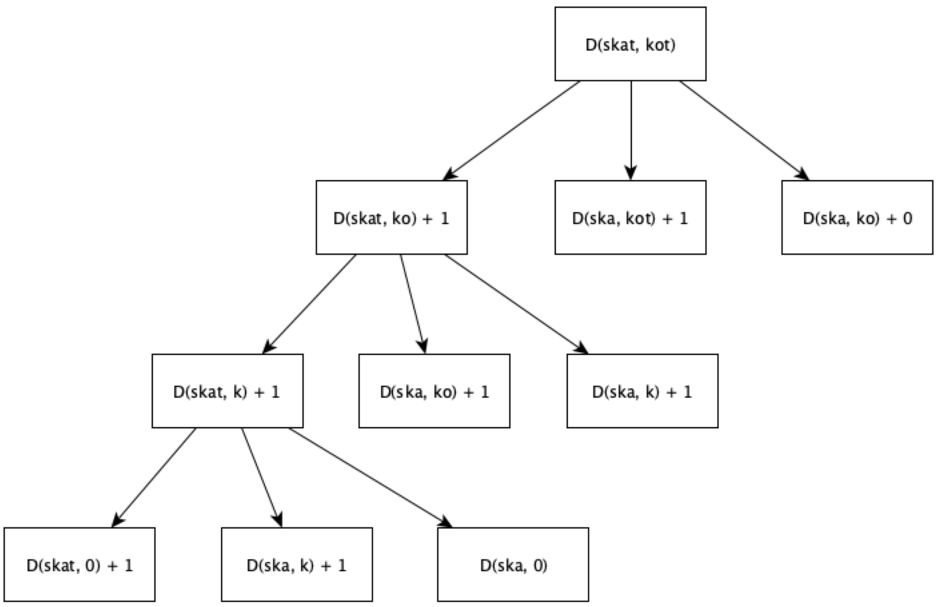

Рис. 7: Схема рекурсивного вызова в алгоритме нахождения расстояния Левенштейна (описывает часть работы алгоритма)

На рисунке 7 видно, что рекурсивная реализация алгоритма поиска расстояния между строками использует многократный вызов функций, причем функции могут вызываться с одинаковыми аргументами. Например, 2 раза вызывается функция с аргументами "ska"и "ko". При табличном способе создается матрица, размерность которой определяется длинной слов, что позволяет оптимизировать кол-во необходимой памяти. В отличии от табличного способа, рекурсивный требует много памяти (кол-во вызовов функций при рекурсии 481, при длине слов: 4 символа). Очевидно, что рекурсивный метод проигрывает табличному по памяти и по скорости, поэтому использование такого метода становится нецелесообразным.

4.4 Вывод

Реализации алгоритма нахождения расстояния Левенштейна и алгоритма нахождения расстояния Дамерау-Левенштейна сравнимы по времени. Худшие результаты измерений показывает рекурсивная реализация алгоритма нахождения расстояния Дамерау-Левенштейна - матричные реализации заметно выигрывают при росте длины строк. Такое замедление по времени вызвано частыми обращениями к рекурсивным запросам. Можно сделать вывод, что данная версия алгоритма не рекомендована к использованию в реальных проектах, так как она может существенно замедлить их работу.

Заключение

В результате выполнения лабораторной работы были получены следующие основные навыки:

1. изучены теоритические понятия в алгоритмах для нахождения расстояния Левенштейна и Дамерау-Левенштейна;
2. проведен аналитический вывод формул для заполнения матриц расстояний;
3. проведено сравнение трех реализаций заданного алгоритма, выявлены их слабые места;
4. в рамках данной работы было сделано заключение, что рекурсивный алгоритм сильно проигрывает по скорости двум другим матричным реализациям;
5. найденные скоростные отличия между матричными реализациями алгоритма Левенштейна и алгоритма Дамерау-Левенштейна крайне малы;
6. определение расстояния по Левенштейну имеет недостатки: a) при перестановке местами слов или частей слов получаются большие расстояния; b) расстояния между совершенно разными короткими словами будут меньше в отличии от расстояния между двумя очень похожими, но длинными словами.

Список литературы

[1] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.

[2] Нечёткий поиск в тексте и словаре // [Электронный ресурс]. Режим доступа: https://habr.com/ru/post/114997/ (дата обращения: 10.09.19).

[3] Вычисление расстояния Левенштейна // [Электронный ресурс]. Режим доступа: https://foxford.ru/wiki/informatika/vychislenie-rasstoyaniya-levenshteyna (дата обращения: 10.09.19).

[4] Нечеткий поиск, расстояние левенштейна алгоритм // [Электронный ресурс]. Режим доступа: https://steptosleep.ru/antananarivo-106/ (дата обращения: 10.09.19).