

Московский государственный университет имени М.В. Ломоносова

Факультет Вычислительной математики и кибернетики

Кафедра Алгоритмических языков

**Библиотека для поддержки трансформационных правил**

Обзорная глава диплома

**Выполнил:**

студент 425 группы

Ступеньков В. О.

**Научный руководитель:**

д.ф-м.н., профессор

Соловьев С. Ю.

Москва, 2022

Содержание

TODO

[Аннотация](#_Toc121847267)

[Введение](#_Toc121847267) 4

[1. Постановка задачи](#_Toc121847268) 6

[2. Обзор существующих решений](#_Toc121847269) 7

[2.1 Основные понятия 7](#_Toc121847271)

[2.2 Расстояние Левенштейна 8](#_Toc121847272)

[2.2.1 Определение и рекуррентная формула 8](#_Toc121847273)

[2.2.2 Алгоритм Вагнера-Фишера 10](#_Toc121847274)

[2.2.3 Метод динамического программирования вычисления структурного сходства Левенштейна 12](#_Toc121847275)

[2.2.4 Расстояние Дамерау-Левенштейна 15](#_Toc121847276)

[2.3 Категориальный синтез 17](#_Toc121847277)

**3. Исследование и построение решения задачи**

**4. Заключение**

[Список литературы: 19](#_Toc121847278)

**Аннотация**

В данной выпускной квалификационной работе разрабатывается библиотека для поддержки создания и использования трансформационных правил. Рассмотрены существующие алгоритмы работы с правилами, изменяющими синтаксическую струкутру строк, в том числе алгоритмы вычисления расстояния Левенштейна и структурного сходства двух строк. Реализована модификация существующего алгоритма вычисления структурного сходства, позволяющая работать со строками, включающими переменные. Реализованы функции, позволяющие использовать эти алгоритмы для создания трансформационных правил. Разработана база правил и алгоритмы разрешения конфликтов, обеспечивающие возможность хранить и получать нужные пользователю трансформационные правила. Также описана структура библиотеки и приведены примеры использования её функций.

# Введение

Задачи, связанные с обработкой текстовых данных, традиционно представляют значительный интерес для исследований. Они имеют широкое применение в различных областях научной деятельности: от информационного поиска до обработки последовательностей генов. В подобных задачах часто встречается необходимость изменять строки в соответствии с некоторыми правилами, изменяющими синтаксическую структуру строки. Такие правила называются трансформационными. Примерами их применения могут служить задачи, связанные с приведением слов к нормальной форме, изменением падежа, числа, склонения слова. Также применение таких правил можно обнаружить в задачах поиска и замены слов, например в поисковых системах. Существует множество библиотек, так или иначе использующих преобразования синтаксической структуры строк для решения описанных задач.

Для работы с трансформационными правилами требуются инструменты, позволяющие создавать, хранить и использовать их. Данная работа посвящена разработке такого инструмента в виде библиотеки, предоставляющей разнообразные методы для работы с трансформационными правилами.

При решении указанных подзадач создания и применения правил возникает необходимость использовать алгоритмы нечёткого поиска, для сопоставления информации заданному образцу поиска или близкому к этому образцу значению. Алгоритмы нечёткого поиска используются во многих современных системах. Одной из реализацией алгоритма нечёткого поиска является расстояние Левенштейна – минимальное количество операций вставки, удаления и замены одного символа на другой, необходимых для превращения одной строки в другую [1].

Другой метод, реализующий алгоритм нечёткого поиска – поиск наибольшей общей подпоследовательности или вычисление структурного сходства двух строк. Он наиболее удобен для применения при создании и применении трансформационных правил. Трансформационное правило можно представить в виде пары строк – левой, являющейся условием применимости правила, и правой, являющейся результатом применения этого правила. При использовании правила условие применимости сопоставляется с входной строкой при помощи алгоритма вычисления структурного сходства. В случае же с созданием правил, при помощи этого алгоритма можно выделить общие части входных строк и построить на их основе некоторые обобщённые правила, то есть правила, включающие переменные, которым при сопоставлении со строкой может быть присвоено некоторое константное значение. Однако алгоритм в известном виде не пригоден для использования с правилами такого вида.

Разработанная в рамках данной работы библиотека включает в себя реализацию алгоритма вычисления структурного сходства Левенштейна, а также его модификацию, позволяющую применять его для строк с переменными. Реализация этих алгоритмов, а также способов их применения для создания и применения трансформационных правил будет описана в следующих разделах.

Трансформационные правила представляют собой частный случай продукционных правил. Действительно, у них, также как у продукций, выделяют условие применимости и результат применения правила. В связи с этим некоторые элементы теории продукционных правил применимы в этой работе. В разработанной библиотеке это используется прежде всего при хранении правил. На множестве трансформационных правил можно определить отношение «общее-частное», теоретические основы его применения для продукций описаны в [2]. Это позволяет определить стратегии для выбора правил из базы данных, которые будут описаны далее.

Прослеживается некоторая связь между трансформационными и продукционными правилами. В некоторой степени, трансформационные правила являются частным случаем правил продукционных. В связи с чем некоторые элементы теории продукций применимы к рассматриваемым правилам. В первую очередь это касается определения применимости правил и их ранжирования. В следующих главах будут описаны методы ранжирования правил, использующиеся в библиотеке.

Несмотря на широту применения трансформационных правил, отсутствуют универсальные библиотеки, дающие возможность создавать, хранить и использовать разнообразные правила для строк, а также экспериментировать с выводом универсальных правил. Целью данной работы является разработка именно такого инструмента.

# 1. Постановка задачи

Основная цель данной работы – создание библиотеки для поддержки работы с трансформационными правилами, подходящей для использования при решении различных задач, связанных с обработкой текстовых данных.

Для этого требуется:

1. На основе существующих алгоритмов вычисления расстояния Левенштейна и поиска наибольшей общей подпоследовательности реализовать алгоритм, удобный для использования при создании и применении трансформационных правил

2. На основе реализованного алгоритма реализовать программную библиотеку, позволяющую предложить пользователю различные варианты для создания и использования трансформационных правил.

3. Создать базу правил для хранения трансформационных правил и отношений между ними

4. Оформить полученные результаты в виде библиотеки на языке Python с удобным интерфейсом.

# 2. Обзор существующих решений

## В данной главе рассмотрены основные понятия и алгоритмы, применяемые при работе с трансформационными правилами. В первую очередь речь идёт о расстоянии Левенштейна и способах его вычисления. Также приведены описание алгоритма нахождения наибольшей общей подпоследовательности, который положен в основу алгоритма вычисления структурного сходства и описание методов ранжирования продукционных правил, используемых в продукционных системах.

## 2.1 Основные понятия

Трансформационное правило – это правило, преобразующее синтаксическую структуру строки. Удаление, вставка и перемещение символов являются примерами трансформационных правил.

Продукции представляют собой операторы специального вида и состоят из двух основных частей, для краткости называемых обычно “ситуация – действие”. “Ситуация” содержит описание ситуации, в которой применима продукция, “действие” – это набор инструкций, подлежащих выполнению в случае применимости продукций.

Нечеткий поиск – это поиск информации, при котором выполняется сопоставление информации заданному образцу поиска или близкому к этому образцу значению.

Образец – строка специального вида, некоторые фрагменты которой заменены переменными. Образцы задают некоторое множество строк – при подстановке вместо переменных образца констант получаются фактические строки.

Под исключением из общего правила в контексте трансформационных правил понимается правило, область определения которого является частным случаем области определения другого правила, которое называется общим правилом.

Системы, с введёнными в них отношениями “исключение – общее правило” называются системами с исключениями.

## 2.2 Расстояние Левенштейна

### 2.2.1 Определение и рекуррентная формула

Рассмотрим определение алгоритма нечёткого поиска, данное выше. Алгоритмы нечеткого поиска используются в большинстве современных поисковых систем, например для проверки орфографии.

Формальное определение алгоритма:

Пусть ∑ – конечное множество (алфавит) размера |∑| = σ.

Пусть T ∈ ∑\* – текст длиной n = |T|.

Пусть P ∈ ∑\* – образец длиной m = |P|.

Пусть k ∈ ℝ – максимально разрешенное количество ошибок.

Пусть d : ∑\* × ∑\* → ℝ – функция расстояния.

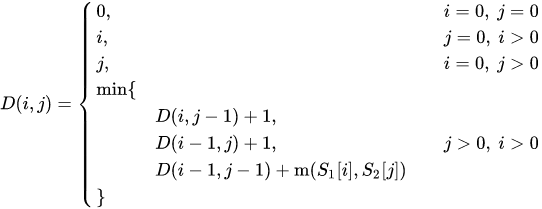
Тогда задача формулируется так: даны T,P,k и d(.), которые возвращают множества всех позиций текста j таких, что существует i такое, что d(P,Ti…j) ≤ k [2].

В упрощенном виде задачу алгоритма можно описать так: по заданному слову найти в тексте c размером n все слова, совпадающие с этим словом (или начинающиеся с этого слова), учитывая k возможных различий (неточностей).

Одним из самых известных примеров практической реализации данного алгоритма является вычисление расстояния Левенштейна.

Расстояние Левенштейна 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.

Рекуррентная формула, описывающая расстояние Левенштейна:

Рисунок 1: Рекуррентная формула для расстояния Левенштейна

Свойствами расстояния Левенштейна являются:

1) d(S1,S2)⩾||S1|−|S2||

2) d(S1,S2)⩽max(|S1|,|S2|)

3) d(S1,S2)=0⇔S1=S2,

где d(S1,S2) – расстояние Левенштейна между строками S1 и S2, а |S| – длина строки S. Также расстояние Левенштейна является метрикой, так как отвечает её свойствам:

1) d(x,y) = 0 <=> x = y (аксиома тождества)

2) d(x,y) = d(y,x) (аксиома симметрии)

3) d(x,z) ⩽ d(x,y) + d(y,z) (неравенство треугольника)

Для вычисления расстояния Левенштейна существует несколько практических подходов, которые будут описаны ниже. Нужно отметить, что в связи с особенностями решаемой задачи алгоритм, применяемый непосредственно в работе, является несколько модифицированной версией классических методов вычисления расстояния Левенштейна.

### 2.2.2 Алгоритм Вагнера-Фишера

Для нахождения редакционного расстояния между двумя строками необходимо вычислить матрицу D, используя рекурсивную формулу, приведённую ранее. Её можно вычислять как по строкам, так и по столбцам. Псевдокод алгоритма описан в [3] при произвольных ценах замен, вставок и удалений. Данный псевдокод решает простой частный случай, когда вставка символа, удаление символа и замена одного символа на другой стоят одинаково для любых символов.

Алгоритм в описанном виде требует O(M\*N) операций и столько же памяти, где M и N – длины входных строк. Если требуется вычислить только расстояние, то алгоритму не требуется памяти больше, чем на два столбца, то есть O(N). Однако матрица в полном объеме нужна для восстановления редакционного предписания, то есть последовательности действий, необходимых для получения из первой строки второй кратчайшим образом.

Для вычисления редакционного предписания требуется вычислить матрицу D, после чего идти из правого нижнего угла (M,N) в левый верхний, на каждом шаге ища минимальное из трёх значений:

* если минимально (D(i-1, j) + 1), добавляем удаление символа S1[i] и идём в (i-1, j);
* если минимально (D(i, j-1) + 1), добавляем вставку символа S1[i] и идём в (i, j-1);
* если минимально (D(i-1, j-1) + С), добавляем замену если С = 1.

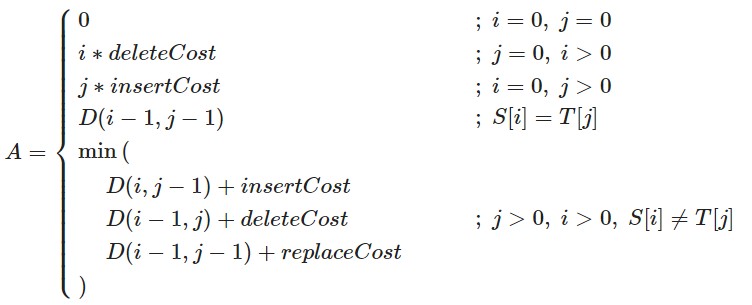
### 2.2.3 Расстояние Дамерау-Левенштейна

Помимо классического расстояния Левенштейна широко распространено использование так называемого расстояния Дамерау-Левенштена.

Расстояние Дамерау-Левенштейна между двумя строками, состоящими из конечного числа символов — это минимальное число операций вставки, удаления, замены одного символа и перестановки двух соседних символов, необходимых для перевода одной строки в другую. Как можно понять из определения, оно является модификацией расстояния Левенштейна, отличающейся от него добавлением операции перестановки.

Расстояние Дамерау-Левенштейна, как и метрика Левенштейна, является мерой "схожести" двух строк. Алгоритм его поиска находит применение в реализации нечёткого поиска, а также в биоинформатике (сравнение ДНК), несмотря на то, что изначально алгоритм разрабатывался для сравнения текстов, набранных человеком.

В основу алгоритма вычисления расстояния Дамерау-Левенштейна положена идея динамического программирования по префиксу, также, как и в методе Вагнера-Фишера. Подробно метод вычисления и его вывод описан в [4]. Основное отличие от вычисления расстояния Левенштейна заключается в рекуррентной формуле:

Рисунок 2: Рекуррентная формула для расстояния Дамерау-Левенштейна

Подход, описанный в предыдущем разделе, очевидно не учитывает перестановки символов и в описанном виде не подходит для реализации вычисления расстояния Дамерау-Левенштейна. Так как в рамках разработки библиотеки ставится цель охватить как можно большее количество методов работы с трансформационными правилами, желательно получить возможность вычислять и этот тип расстояния Левенштейна.

**2.3 Структурное сходство Левенштейна**

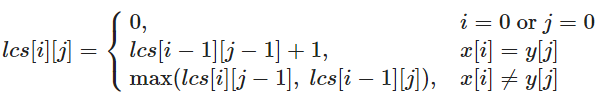
**2.3.1 Поиск наибольшей общей подпоследовательности**

Поиск наибольшей общей подпоследовательности сосредотачивается на поиске сходства между строками, в отличии от расстояния Левенштейна. Основные теоретические сведения о задаче наибольшей общей подпоследовательности и способах её решения приведены в [6]. В этом разделе описаны особенности применения к строкам алгоритма динамического программирования решения этой задачи.

В качестве последовательностей рассматриваются строки X и Y, состоящие из символов (x1, x2, …, xm) и (y1, y2,…, yn).

Обозначим как lcs[i][j] наибольшую общую подпоследовательность префиксов данных строк, заканчивающихся в элементах с номерами i и j соответственно. Таким образом, для вычисления lcs потребуется заполнить матрицу D(m,n) при помощи следующего рекуррентного соотношения:

Сложность данного алгоритма O(m\*n), где m и n – длины строк.

Рисунок 3: Рекуррентная формула для алгоритма поиска наибольшей общей подпоследовательности

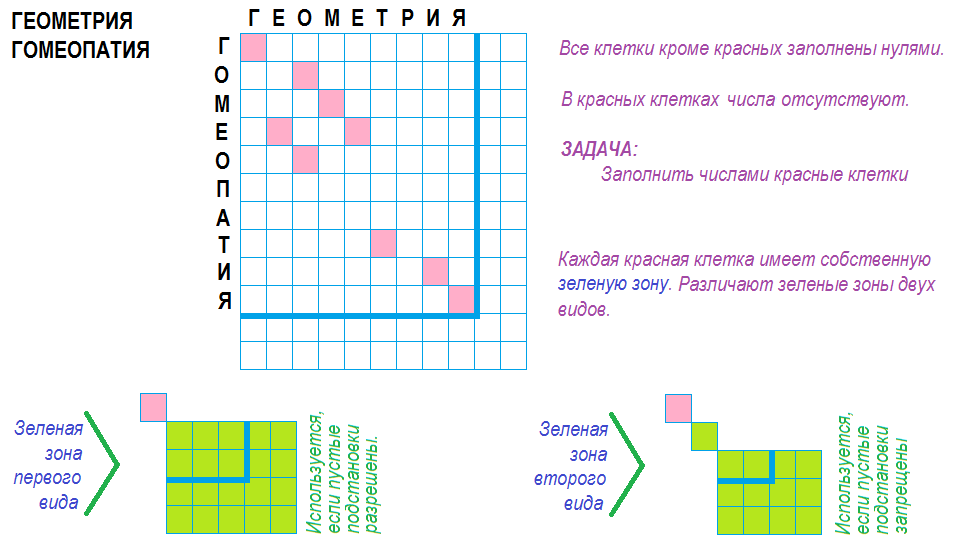
Для каждой пары символов строк помимо длины наибольшей общей подпоследовательности соответствующих префиксов хранятся также номера последних элементов, входящих в эту подпоследовательность. Таким образом, посчитав матрицу по рекуррентной формуле, можно восстановить всю общую подпоследовательность.

**2.3.2 Алгоритм вычисления структурного сходства Левенштейна**

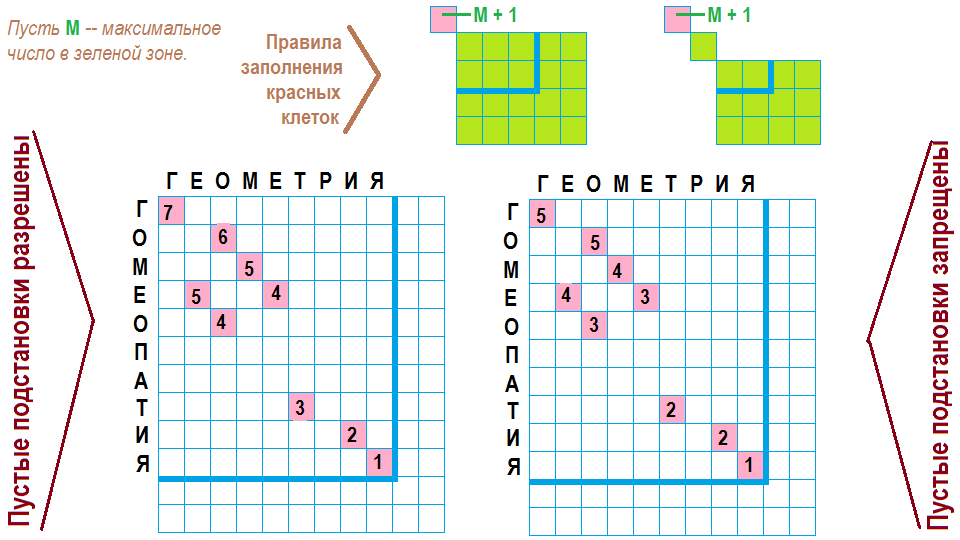
Функции разрабатываемой нами библиотеки предполагают использование некоторого алгоритма, подходящего как для сопоставления текста образцу, так и для генерации новых правил на основе входных текстовых данных. Соответственно, используемый алгоритм должен не только вычислять расстояние Левенштейна между двумя строками, но и создавать на основе вычисленной матрицы результирующую строку, являющуюся образцом, которому соответствуют обе входные строки.

Для этого предлагается следующий подход к вычислению структурного сходства Левенштейна, который продемонстрирован на иллюстрациях к конкретному примеру. Результирующая матрица D заполняется следующим образом: все поля матрицы, кроме тех, что находятся на пересечении одинаковых символов, заполняются нулями. Для каждой ненулевой клетки матрицы, выделяется подматрица матрицы D. В этой подматрице все клетки считаются вычисленными. Нужно отметить, что данный подход подразумевает два разных случая: пустые подстановки при вычислении расстояния Левенштейна могут быть разрешены и запрещены.

Главное отличие при вычислении расстояния заключается в том, что подматрицы выделяются по-разному, что можно увидеть на иллюстрации работы алгоритма.

Рисунок 4: Вычисление структурного сходства Левенштейна: вид матрицы

Среди значений, находящихся в указанной подматрице, находится максимум, к которому прибавляется единица. Полученное значение записывается в выделенную клетку. Выделенные клетки, для которых не находится ненулевых значений, заполняются единицами. Этот процесс продолжается до тех пор, пока матрица не будет заполнена.

Рисунок 5: Вычисление струкутрного сходства Левенштейна: заполнение матрицы

Такой подход позволяет избежать необходимости вычислять каждую клетку матрицы и позволяет находить структурное сходство Левенштейна наглядным способом. Заполненные ненулевые клетки образуют граф, максимальный путь в котором может быть найден динамическим проходом по матрице. Результатом этого прохода является строка, включающая общие части поданных на вход строк и переменные, которым может быть присвоено одно из двух константных значений.

На следующей иллюстрации изображены полученные в результате работы алгоритма строки. Как видно на этом примере, длина максимального пути, количество и значения переменных могут отличаться в зависимости от того, разрешены ли пустые подстановки. В практической части данной работы реализованы оба метода, чтобы позволить пользователю библиотеки выбирать подходящий для его задачи способ вычисления структурного сходства Левенштейна.

Рисунок 6: Вычисление структурного сходства Левенштейна: создание образцов

Главная особенность этого алгоритма – его обобщающая способность, позволяющая выделять образцы из двух строк. Он основан на уже описанном алгоритме вычисления наибольшей общей подпоследовательности, но в отличии от него, модифицирован таким образом, чтобы быть удобным для работы с трансформационными правилами.

## 2.4 Продукционные правила

Выше уже приводилось определение продукционных правил. Несложно заметить, что трансформационные правила являются частным случаем продукционных правил: левая часть является их условием применимости, а правая – результатом применения правила. Поэтому решения, существующие для продукций могут быть применены для работы с трансформационными правилами. Далее приведены некоторые теоретические сведения, связанные с продукционными правилами, применение которых в разработанной библиотеке будет описано в следующей главе.

**2.4.1 Категориальный синтез**

В работе [2] представлены основные тезисы, связанные с исключениями и их использованием в программных системах. Ниже представлено определение и обоснование использования категориального синтеза для вывода обобщённых правил на основе существующих правил.

В программировании широко распространён принцип, согласно которому необходимо соблюдать определённый порядок следования операторов с условной частью, при котором частные правила должны располагаться перед общими. Отсюда возникает необходимость определить отношение “общее – частное” на множестве условий применимости правил – правила с более частными условиями пользуются приоритетом.

Для двух правил с константной условной частью «дом → дома» и «лом → лома» может быть выведено обобщённое правило:

«X → Xа»

Это правило тесно связано с породившими его правилами и непосредственно переходит в них при подстановке констант вместо переменных. Таким образом возникает задача получения образца, то есть обобщённого правила специального вида на основе поступивших правил. Примерами использования подобных образцов являются, в частности, подстановки алгоритмов Маркова и операторы языка Рефал.

Образец, полученный из исходных правил, называется полученным посредством категориального синтеза и является обобщённым оператором. Обобщённые операторы характеризуются тем, что они содержат переменные, вместо которых можно подставлять конкретные константы. В этом и состоит применение оператора.

Таким образом, категориальный синтез позволяет при обобщении использовать целую систему правил и приводит к построению весьма выразительных обобщённых операторов. Категориальный синтез и подстановочное обобщение вводят на множестве правил отношение “общее – частное”.

В общем случае в системах с исключениями распознавание общих операторов и исключений производится следующим образом. Если для двух операторов p и q область определения оператора p является частным случаем области определения оператора q, но обратное неверно, то оператор p является исключением из оператора q, а q – общее правило для p.

Детали реализации аппарата исключений на множестве трансформационных правил, построенного в рамках этой задачи, будут описаны в следующих главах.

**2.4.2 Алгоритмы разрешения конфликтов**

При реализации вывода в продукционных системах левые части правил, то есть их условия применимости, сопоставляются с базой данных и те из них, условия применимости которых выполнены, добавляются в конфликтное множество. Конфликтное множество представляет собой множество всех продукционных правил, условия которых выполнены, но которые ещё не были выполнены. Для выбора правила из конфликтного множества могут применяться различные стратегии, называемые алгоритмами разрешения конфликтов.

Далее приведены описания алгоритмов разрешения конфликтов, применявшихся при разработке библиотеки.

* Стратегия «простоты»: среди применимых правил выбираются правила с наименьшей специфичностью. Специфичность правила определяется числом сравнений, которые должны быть выполнены в левой части правила.
* Стратегия «сложности»: среди применимых правил выбираются правила с наибольшей специфичностью. В терминах отношений «общее-частное» в этой стратегии выбирается наиболее частное правило, а в предыдущей – наиболее общее.
* Стратегия MEA: выбор правил основан в первую очередь на временном теге образца, связанным с первым условием в правиле. Выбирается правило, временной тег которого больше тегов других правил.
* Стратегия случайного выбора: каждому применимому правилу сопоставляется случайное число, которое используется для определения его положения в конфликтном множестве. Выбирается правило с наибольшим числом.

Также каждому правилу сопоставляется значение его значимости или приоритет. Он учитывается при составлении конфликтного множества: выбираются правила с наибольшими приоритетами, если не указано обратное.

**3 Исследование и построение решения задачи**

Как было указано ранее, для решения задач создания и использования трансформационных правил необходимо использовать алгоритмы нечёткого поиска. Наиболее подходящим для поставленных в этой работе целей является алгоритм вычисления структурного сходства Левенштейна. Однако в описанном виде его не получится применить к правилам, содержащим переменные. Поэтому в рамках данной работы была разработана модификация алгоритма вычисления структурного сходства Левенштейна, подходящая для работы со строками с переменными частями.

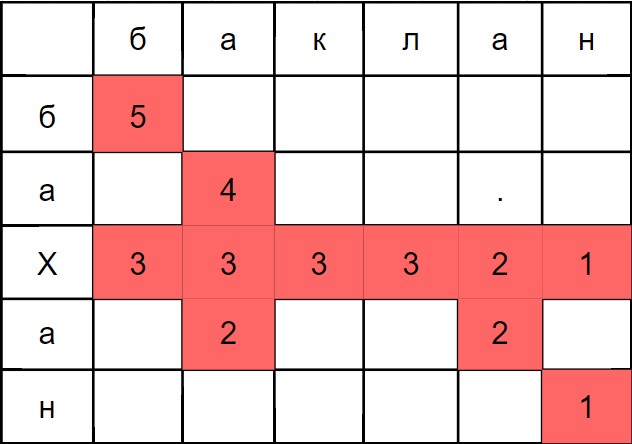
В данном разделе описана реализация этой модификации, её использование для решения задач создания трансформационных правил и сопоставления образцов.

Помимо этого, в этой части описаны реализация аппарата исключений на множестве трансформационных правил и методы ранжирования трансформационных правил из базы данных, основанные на соответствующих методах для продукций, приведённых в предыдущей главе.

**3.1 Модификация алгоритма вычисления структурного сходства**

Используемый в работе алгоритм является модификацией алгоритма вычисления структурного сходства, который был описан во втором разделе. Основным отличием разработанной в рамках данной работы модификации является возможность работать со строками, включающими переменные. Реализации оригинального алгоритма и его модификации находятся в репозитории [8], содержащем библиотеку.

Для реализации этой возможности изменяется способ заполнения матрицы D: все строки матрицы, соответствующие переменным в первой строке, помечаются как ненулевые и заполняются по тем же принципам, что и в методе вычисления структурного сходства:

Рисунок 7: Заполнение матрицы для модифицированного алгоритма вычисления структурного сходства Левенштейна

В результате применения алгоритма, в матрице может получится несколько эквивалентных по длине путей. В целях нахождения всех максимальных подпоследовательностей и построения образцов для каждой из них используется следующий алгоритм. Для каждой позиции в матрице, содержащей максимальное значение, создаётся строка с результатом, и производится первый шаг построения образца. После чего для каждого образца последовательно ищутся значения максимальных позиций в соответствующей подматрице. Для каждой найденной на текущем шаге позиции, содержащей максимальное значение, делается очередной шаг построения образца и производится поиск следующих позиций с максимальными значениями. Процесс продолжается до тех пор, пока все образцы не будут достроены до конца.

Этот алгоритм можно описать следующим псевдокодом:

**Исходные данные:** строки s1 и s2, между которыми вычисляется структурное сходство, заполненная матрица D.

**Результат:** словарь, содержащий образцы с множеством заполненных переменных

results = {}

**для каждого** b ∈ Begins **выполнить**

prev\_i := 0

prev\_j := 0

max\_i := 0

max\_j := 0

vars = {}

samples = {}

first\_step = build\_rule\_step(" ", s1, s2, prev\_i, prev\_j, vars, b) // выполнение первого шага построения образца

samples[first\_step] = [b, tmp\_vars]

**пока** max\_i < n **и** max\_j < m **выполнять**

tmp\_dict = {}

**для каждого** key ∈ samples **выполнять**

prev\_i := samples[key][0][0]

prev\_j := samples[key[0][1]]

maxes := find\_max\_poses(D[prev\_i+1:][prev\_j+1:], n, m, blanks)

**для каждого** max ∈ maxes **выполнять**

step = build\_rule\_step(key, s1, s2, prev\_i, prev\_j, vars)

samples[step] = [[max[0], max[1]], vars]

**конец цикла**

**конец цикла**

**для каждого** key ∈ samples **выполнять**

results[key] = samples[key]

**конец цикла**

Здесь D – матрица, заполненная описанным в этом подразделе способом,

results, samples – словари, содержащие образцы и заполненные переменные,

begins – множество позиций матрицы с максимальным значением,

vars – словарь, содержащий переменные и их значения

build\_rule\_step – функция, возвращающая образец, полученный в результате совершения одного шага при проходе матрицы. Она анализирует символы входных строк на текущем шаге и вставляет в образец по мере необходимости переменные, одновременно заполняя словарь vars, который передаётся в функцию по ссылке. В процессе необходимо проверять относительное расположение позиций, рассматриваемых на данном шаге, и позиций, рассмотренных на предыдущем. Также необходимо проверять, является ли рассматриваемый символ в первой строке переменной, чтобы обеспечить корректное добавление символов в образец и заполнение переменных значениями.

Эту функцию можно описать следующим псевдокодом:

build\_rule\_step(result, s1, s2, prev\_i, prev\_j, vars, max):

i := max[0]

j := max[1]

**Если** i == prev\_i **или** j == prev\_j **то**

**Если** i != 0 **или** j != 0 **то**

Добавление новой переменной X в vars

result += X

**Если** s1[i] является переменной **то**

Дописать в последнюю переменную или добавить новую переменную в vars и в result

**Иначе**

result += s2[j]

**Иначе Если** i == prev\_i + 1 **и** j == prev\_j + 1 **то**

**Если** s1[i] является переменной **то**

Дописать в последнюю переменную или добавить новую переменную в vars и в result

**Иначе**

result += s2[j]

**Иначе Если** i != prev\_i+1 **или** j != prev\_j **то**

Дописать в последнюю переменную или добавить новую в vars и в result

**Если** s1[i] является переменной **то**

Дописать в последнюю переменную

**Иначе**

result += s2[j]

return result

Основное отличие этого алгоритма в необходимости динамического построения нескольких строк, связанного с тем, что при наличии переменных в обеих входных строках, количество различных максимальных путей в матрице D значительно увеличивается. Также при построении образца возникает необходимость дополнительно проверять, являются ли символы, рассматриваемые на текущем шаге, переменными.

**3.2 Использование алгоритма вычисления структурного сходства Левенштейна и его модификации**

Реализация описанного в предыдущей главе алгоритма используется в разработанной библиотеке для решения нескольких задач, перечисленных ниже.

1) Создание трансформационных правил на основе двух строк. Как уже упоминалось, основным достоинством алгоритма вычисления структурного сходства является его обобщаю способность, что делает его крайне полезным для решения этой задачи. Сформулировать эту задачу можно следующим образом: необходимо на основе двух входных строк создать трансформационные правила, позволяющие преобразовывать строки, подобные первой строке, в строки, подобные второй строке.

Делается это следующим образом: на основе алгоритма вычисления структурного сходства строится образец, включающий переменные, которому соответствуют обе входные строки. Для примера, приведённого во втором разделе данной работы одним из таких образцов является строка «гXмеYия». Переменные в данном случае инициализированы следующими парами значений: X: («ео», «о»), Y: («тр». «опат»). Из полученного образца можно составить следующее трансформационное правило: Y1еоY2трY3 → Y1оY2опатY3.

2) Проверка на соответствие строки образцу. Эта задача широко распространена и существует множество способов её решения. В разработанной библиотеке реализация её решения добавлена в основном для удобства пользователя. При сопоставлении строки с образцом используется модификация алгоритма вычисления структурного сходства. Если строка соответствует образцу, то в результате применения этого алгоритма одна из построенных строк будет совпадать с заданным образцом, а переменные, входящие в состав образца, будут заполнены константными частями входной строки. Например, при проверке соответствия строки «баклан» образцу «баXан», в результате работы алгоритма будет выведена строка «баXан», а переменная будет иметь значение X = «кл»

3) Применение правила к входной строке. Очевидно, реализация решения этой задачи необходима для разрабатываемой библиотеки. Её решение основано на описанном выше сопоставлении строки образцу. Важной деталью является то, что при сопоставлении в системе записываются значения переменных, полученные в результате работы алгоритма.

Тогда при наличии в базе правил трансформационного правила «Xа –> Xов» и поступлении на вход строки «барабана», условие применимости в левой части правила будет сопоставлено со строкой и переменная X примет значение «барабан». Это значение затем будет подставлено в правую часть правила, возвращая пользователю строку «барабанов».

4) Выделение общих правил на основе существующих. Её можно сформулировать следующим образом: необходимо на основе двух существующих трансформационных правил построить более общее, относительно них, правило. Для решения этой задачи также используется модификация алгоритма вычисления структурного сходства. Происходит это следующим образом: модифицированный алгоритм применяется к левым и правым частям правила, после чего полученные образцы попарно объединятся в новые трансформационные правила. Например, при наличии в базе правил следующих правил: «XомеYия –> XомеYий» и «гXмеYия –> гXмеYий» будет выведено и добавлено общее правило «XмеYия –> XмеYий».

Также вычисление структурного сходства находит своё применение в части функций, отвечающих за выбор наиболее подходящих правил, получаемых из базы. Более подробно это описано в следующем подразделе.

**3.3 Хранение трансформационных правил**

**3.3.1 База правил**

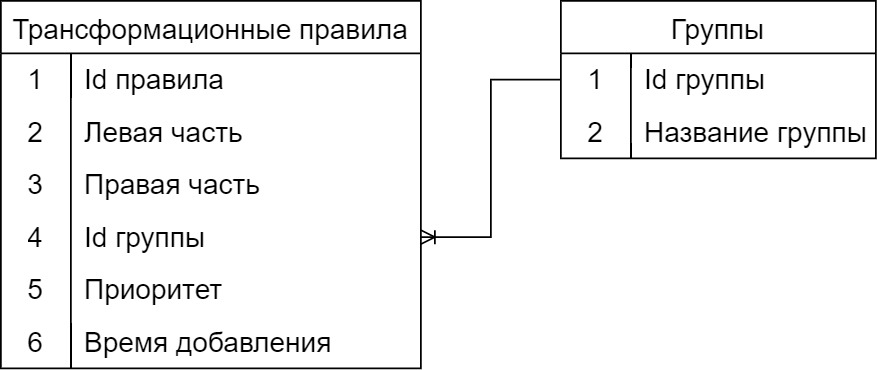
Для хранения трансформационных правил в библиотеке предусмотрена возможность создавать локальную базу данных. Также в библиотеке реализованы методы, позволяющие добавлять, получать, изменять и удалять правила из базы правил.

Помимо самих правил, которые хранятся в базе данных в виде левой и правой части правила, в ней хранятся дополнительные сведения, обеспечивающие удобные способы получения нужных правил. Для каждого правила хранится его приоритет, время добавления, а также группа, к которой оно принадлежит. Остановимся подробнее на каждом из этих полей:

* *Приоритет* является необязательным для заполнения полем и может использоваться пользователем для того, чтобы помечать наиболее важные правила, как те, которым стоит отдавать предпочтение при выборе применяемого правила;
* *Время добавления* автоматически проставляется для каждого из правил и является временным тегом, позволяющим определять, какие правила добавлены раньше или позже всего;
* *Группа* также является необязательным для заполнения полем. Оно даёт возможность пользователю разделять правила на группы и указывать при получении правил из базы, правила из какой группы ему нужны.

База правил находится в третьей нормальной форме и состоит из двух таблиц: «Трансформационные правила» и «Группы».

**TODO Проверить схему**



Как указано на схеме, таблицы находятся в отношении «Один ко многим». В библиотеке реализованы методы, позволяющие указывать приоритеты и группы при создании новых правил, а также при получении правил из базы.

Однако при работе с трансформационными правилами могут возникать ситуации, при которых выполнено условие применимости нескольких правил с одним приоритетом и входящих в одну группу. В таком случае необходимо иметь некоторое отношение между правилами, позволяющее выбрать нужное правило из конфликтного множества. В следующих подразделах описана реализация такого отношения и стратегий его использования в разработанной библиотеке.

**3.3.2 Реализация аппарата исключений на множестве трансформационных правил**

Как уже упоминалось в обзорном разделе, некоторые теоретические положения, справедливые для продукционных правил можно успешно применить для трансформационных правил. Поэтому в разработанной библиотеке на множестве трансформационных правил введено отношение «общее – частное».

В нашей библиотеке желательно, чтобы при поступлении двух правил с константными условными частями «дом → дома» и «лом → лома» было выведено и добавлено обобщённое правило:

«X → Xа»

Это правило тесно связано с породившими его правилами и непосредственно переходит в них при подстановке констант вместо переменных. Таким образом, оно является общим по отношению к породившим его правилам, а эти правила являются частными случаями выведенного правила.

Для определения общих правил в разработанной библиотеке используется модификация алгоритма вычисления структурного сходства Левенштейна. Это происходит следующим образом: левые части правил подаются на вход алгоритму, после чего анализируются построенные образцы. Если среди результатов есть образец, совпадающий с одной из двух входных строк, то условие применимости второй строки является частным случаем условия применимости первой строки. Таким образом, первое правило будет считаться общим для второго, а второе – частным по отношению к первому.

**3.3.3 Алгоритмы разрешения конфликтов**

Описанная реализация определения отношения «общее – частное» на множестве трансформационных правил позволила реализовать различные алгоритмы разрешения конфликтов при получении правил из базы правил.

База правил, реализованная в разработанной библиотеке, не является продукционной системой, однако из-за описанного сходства трансформационных правил с продукциями, алгоритмы разрешения конфликтов, применяемые в таких системах, можно с некоторыми изменениями применить при выборе трансформационных правил из базы.

Таким образом, на основе описанных во втором разделе алгоритмов разрешения конфликтов в разработанной библиотеке были реализованы следующие стратегии:

* Стратегия «простоты»: среди трансформационных правил, условия применимости которых выполнены, выбираются наиболее общие правила.
* Стратегия «сложности»: среди трансформационных правил, условия применимости которых выполнены, выбираются наиболее частные правила.
* Стратегия с временными метками: выбор правила среди применимых трансформационных правил основан в первую очередь на временном теге, который проставляется для каждого правила при его добавлении в базу данных. В библиотеке реализована возможность выбирать как правила с наибольшими временными тегами, так и с наименьшими.
* Стратегия случайного выбора: среди трансформационных правил, условие применимости которых выполнено, выбирается случайное правило, которое и возвращается пользователю.

При этом во всех стратегиях учитываются указанные пользователем приоритеты и группы: среди правил конфликтного множества предпочтение отдаётся правилам с наибольшим приоритетом, а в случае, если пользователь указал номер группы, в конфликтное множество попадают только правила из указанной группы.

**4 Описание практической части**

В данном разделе описаны программные инструменты, использованные при создании библиотеки, а также представлена структура разработанной библиотеки, её основные модули и их назначение. Помимо этого представлены примеры использования библиотеки, описаны тесты и эксперименты, проведённые в рамках практической реализации.

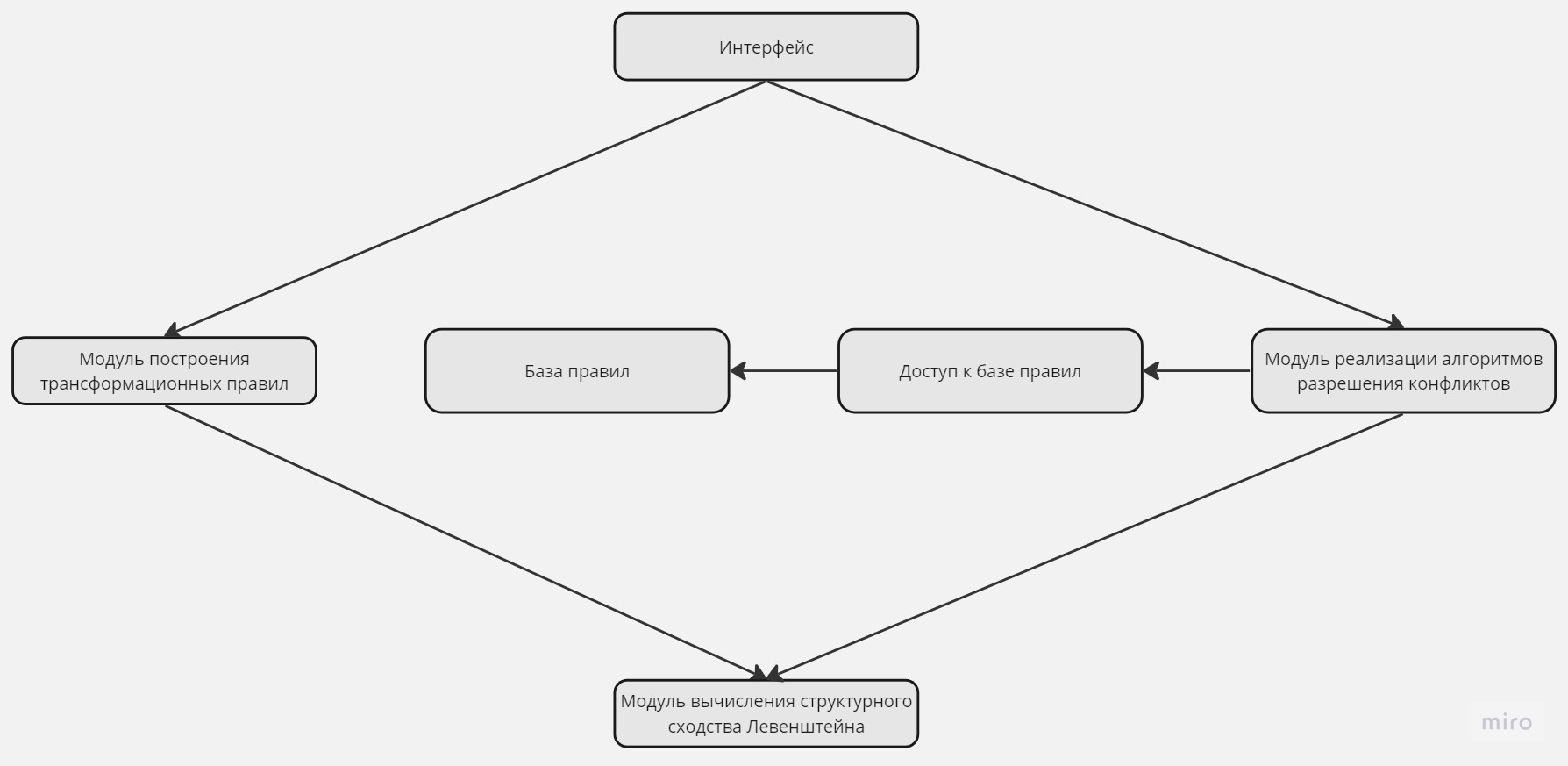
**4.1 Обоснование выбранного инструментария**

В рамках данной работы была реализована библиотека для создания, хранения и использования трансформационных правил. Были реализованы алгоритм вычисления структурного сходства Левенштейна и его модификация, позволяющая находить структурное сходство для строк с переменными, а также методы их использования для работы с трансформационными правилами. Реализация библиотеки проводилась на языке Python 3.9. Одной из причин выбора Python в качестве языка реализации является существование для него большого количества библиотек в области работы со строковыми данными.

Для построения и заполнения матриц использовалась библиотека numpy, являющаяся эффективным инструментом для решения подобных задач. Для создания базы данных и её дальнейшего использования применялись библиотеки sqlite3 и peewee.

**4.2 Структура библиотеки**

На рисунке изображена структура разработанной библиотеки.

Рисунок 8: Структура разработанной библиотеки

В модуле вычисления структурного сходства dynamic\_levenstein находится реализация соответствующего алгоритма и его модификации, позволяющие выделять образцы на основе входных строк.

Модуль построения трансформационных правил use\_levenstein содержит методы для создания правил на основе входных строк и существующих правил с использованием алгоритмов вычисления структурного сходства. Помимо этого, в данном модуле реализованы функции для проверки соответствия строки образцу, то есть левой части трансформационного правила, а также методы позволяющие применять правила к входной строке.

Модуль реализации алгоритмов разрешения конфликтов range\_rules содержит реализацию описанных в предыдущем разделе стратегий выбора трансформационных правил из базы правил. Он также использует реализацию вычисления структурного сходства для определения отношений между правилами.

Отдельным модулем bd реализована работа с базой правил. В нём находятся функции, реализующие получение, добавление, изменение и удаление правил из базы данных. Также в данном модуле присутствуют функции, позволяющие инициализировать, очищать и удалять локальную базу данных.

Сама база правил состоит из двух таблиц «Трансформационные правила» и «Группы», что было описано в предыдущем разделе. Запросы к ней реализованы при помощи библиотек языка Python, указанных в предыдущем подразделе.

Модуль пользовательского интерфейса interface содержит функции, которые доступны пользователю. Они скрывают всю внутреннюю логику работы библиотеки, оставляя пользователю возможность использовать удобные методы с необязательными параметрами для работы с трансформационными правилами.

Далее представлены таблицы с интерфейсами и описанием основных функций, с которыми может работать пользователь. Они разделены на две группы: создание и применение правил и хранение правил.

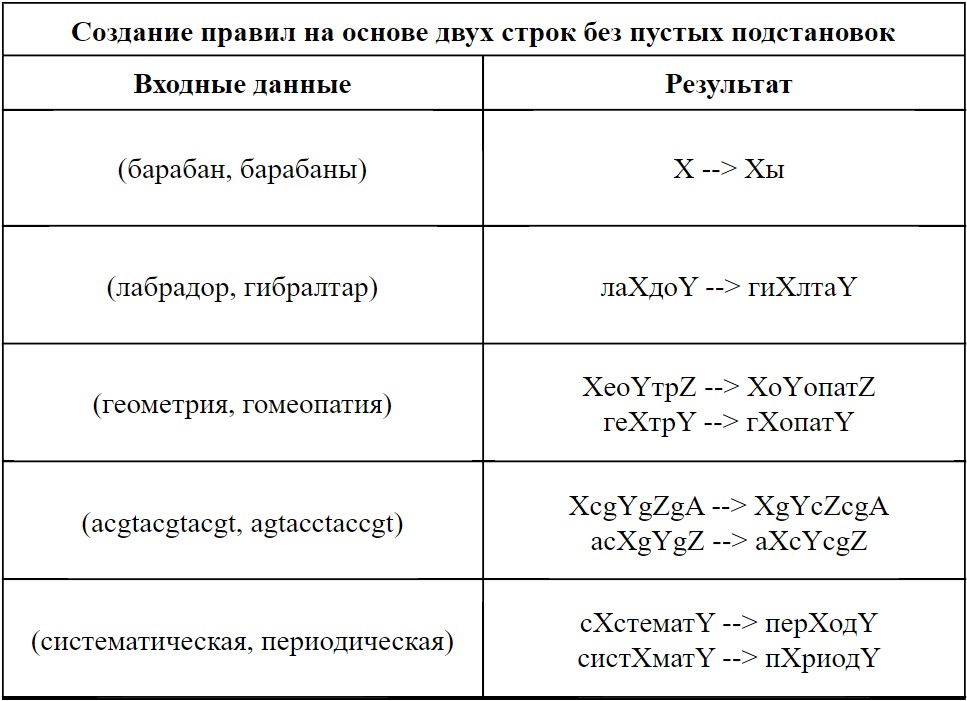
Рисунок 9: Интерфейсы пользовательских функций: создание и использование правил

Рисунок 10: Интерфейсы пользовательских функций: хранение правил

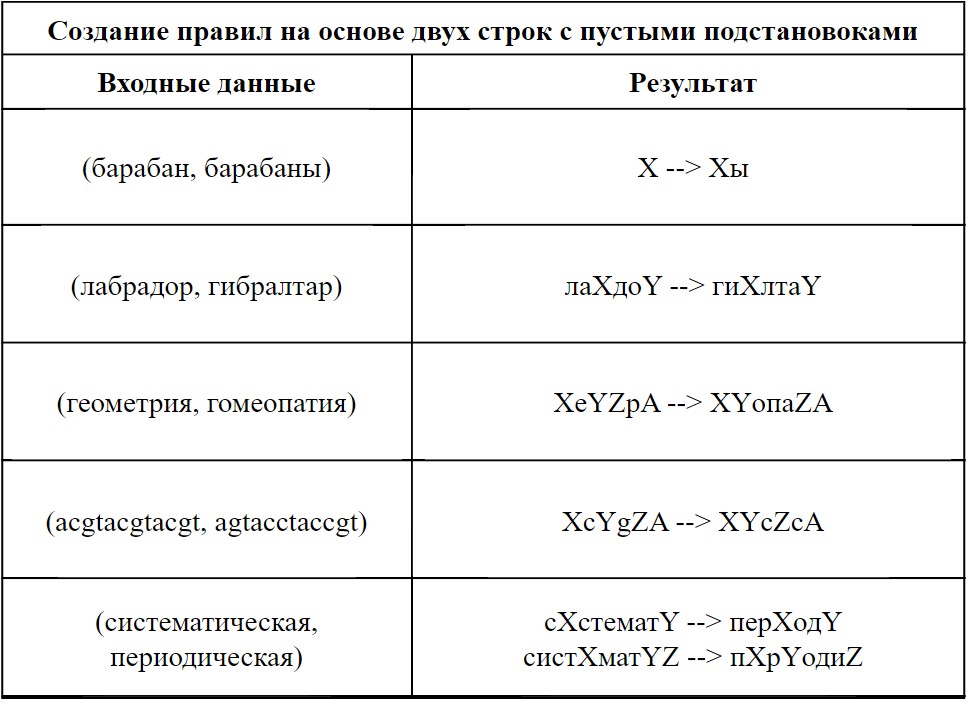
**4.3 Тестирование функций библиотеки**

В данном подразделе приведены примеры использования функций библиотеки для создания трансформационных правил. Для удобства, результаты представлены в таблицах, соответствующих различным способам создания новых правил.

В таблице представлены примеры входных строк и выведенные для них трансформационные правила.

Таблица 1: Примеры использования функций библиотеки для создания трансформационных правил, не разрешающих пустые подстановки

Для сравнения в следующей таблице представлены трансформационные правила, выведенные для тех же строк, но с разрешёнными пустыми подстановками.

Таблица 2: Примеры использования функций библиотеки для создания трансформационных правил, разрешающих пустые подстановки

По приведённым результатам можно увидеть наглядную разницу между работой реализованного алгоритма вычисления структурного сходства при разных параметрах.

В следующей таблице представлены результаты применения модифицированного алгоритма вычисления структурного сходства к образцам, построенным в процессе работы библиотеки на предыдущих примерах.

**TODO таблица**

Также при разработке библиотеки были созданы тесты для тестирования её функций. Они проверяют корректность возвращаемых результатов основных функций пользовательского интерфейса, а также корректность работы функций, реализующих описанные ранее алгоритмы.

**5 Заключение**

В рамках данной работы были изучены методы разработана библиотека, поддерживающая создание, хранение и применение трансформационных правил, которую можно применять при решении различных задач, связанных с обработкой текстовых данных.

В процессе работы были выполнены следующие задачи:

1. Изучены существующие алгоритмы вычисления расстояния Левенштейна и поиска структурного сходства. Реализована модификация алгоритма, удобная для использования в процессе создания и применения трансформационных правил.

2. Создана библиотека, предлагающая пользователю различные функции для создания трансформационных правил, а также для их хранения и применения.

3. Создана база правил, а также реализованы функции, позволяющие выбирать хранящиеся в ней правила при помощи различных стратегий разрешения конфликтов.

4. Результаты оформлены в виде библиотеки на языке Python.

Таким образом, в работе была разработана библиотека для использования трансформационных правил и описаны способы её применения. Главной особенностью реализованной модификации классического алгоритма вычисления структурного сходства, является использование его обобщающей способности для создания образцов, способоных служить основой для обобщённых трансформационных правил. Помимо основных целей работы, были также реализованы различные алгоритмы, дающие возможность работать со сравнением и преобразованием строк.

# Список литературы:

1. В. И. Левенштейн, Двоичные коды с исправлением выпадений, вставок и замещений символов, Докл. АН СССР, 1965, том 163, номер 4, 845–848

2. Кузнецов В. Е. «Представление в ЭВМ неформальных процедур: продукционные системы» М.: Наука. Гл. ред. физ.-мат. лит. – 1989.

3. Задача о редакционном расстоянии, алгоритм Вагнера–Фишера [Электронный ресурс]. – URL: http://neerc.ifmo.ru/wiki/index.php?title=Задача\_о\_ редакционном\_расстоянии,\_алгоритм\_Вагнера–Фишера

4. Задача\_о\_расстоянии\_Дамерау-Левенштейна [Электронный ресурс] – URL https://neerc.ifmo.ru/wiki/index.php?title= Задача\_о\_расстоянии\_Дамерау-Левенштейна

5. Riemsdijk, H. van and E. Williams 1986. *Introduction to the theory of grammar,* MIT Press, Cambridge, Mass.

6. Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн Алгоритмы: построение и анализ — 2-е изд. — М.: «Вильямс», 2007. — с. 459. — ISBN 5-8489-0857-4

7. Hirschberg, D.S.: A linear space algorithm for computing maximal common subsequences. Commun. ACM 18, 341–343 (1975)

**8. Код : https://github.com/ValeryStupenkov/Transformation\_rules**

9. *Гаврилова Т. А., Хорошевский В. Ф.* Базы знаний интеллектуальных систем. Учебник. — СПб.: Питер, 2000