ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ **«КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЛЕКТУАЛЬНЫХ СИСТЕМ

ОТЧЕТ ПО ТЕМЕ:

**«Вычисление расстояния Левенштейна»**

Выполнил: Бекмансуров Илья, гр. 11-102

**Казань, 2022**

**Название алгоритма. Краткая историческая справка.**

Расстояние Левенштейна (редакционное расстояние) – это минимальное число односимвольных операций (а именно вставки одного символа, удаления одного символа и замены одного символа на другой), необходимых для превращения одной последовательности символов в другую. Другими словами, вычисление расстояния Левенштейна – это алгоритм, позволяющий определить «схожесть» двух строк.

Впервые задачу поставил в 1965 году советский математик Владимир Левенштейн при изучении последовательностей 0 – 1. Впоследствии более общую задачу для произвольного алфавита связали с его именем.

**Основной принцип устройства. Особенности.**

Пусть имеются две строки S1 и S2. Мы хотим перевести одну в другую (без ограничения общности будем считать, что мы хотим перевести S1 в S2, так как операции симметричны), используя следующие операции:

* вставка символа в произвольное место
* удаление символа с произвольной позиции
* замена символа на другой

Тогда минимальное количество вышеперечисленных операций для перевода S1 в S2 называется *редакционным расстоянием*, а последовательность проделанных операций - *редакционным предписанием*.

Рассмотрим пример.

Пусть у нас есть строка «скрипкал иса», и мы хотим понять, на что она больше похожа: на «скрип колеса» или «скрипка лиса». Для этого вычислим расстояние Левенштейна (редакционное расстояние).

1. *«скрипкал иса» - «скрип колеса»*

Вставим пробел после «п» -> заменим «о» на «а» -> удалим пробел после «л» -> заменим «и» на «е». Итого, расстояние Левенштейна между последовательностями «скрипкал иса» и «скрип колеса» равно четырем.

*(скрипкал иса -> скрип кал иса -> скрип кол иса -> скрип колиса -> скрип колеса)*

1. *«скрипкал иса» - «скрипка лиса»*

Вставим пробел после «а» -> удалим пробел после «л». Итого, расстояние Левенштейна между последовательностями «скрипкал иса» и «скрипка лиса» равно двум.

*(скрипкал иса -> скрипка л иса -> скрипка лиса)*

Таким образом, получается, что последовательность «скрипкал иса» больше всего «схожа» с последовательностью «скрипка лиса».

Перейдем к реализации алгоритма.

Для нахождения редакционного расстояния цены ранее упомянутых операций (удаления, вставки и замены) могут зависеть от вида операции и/или от участвующих в ней символов. В общем случае:

* w(a, b) — цена замены символа a на символ b
* w(ε, b) — цена вставки символа b
* w(a, ε) — цена удаления символа a

Расстояние Левенштейна является частным случаем задачи нахождения редакционного расстояния при

* w(a, а) = 0
* w(a, b) = 1 при a ≠ b
* w(ε, b) = 1
* w(a, ε) = 1

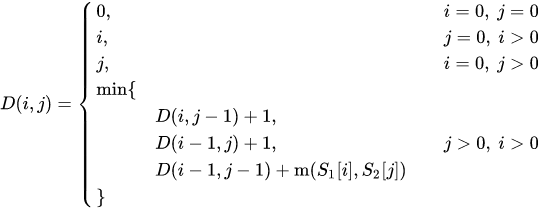
Помимо этого, мы считаем, что цены всех операций неотрицательны, и действует неравенство треугольника: замена двух последовательных операций одной не увеличит общую цену (например, замена символа x на y, а потом y на z не лучше, чем сразу x на z).

Так как же вычисляется расстояние Левенштейна?

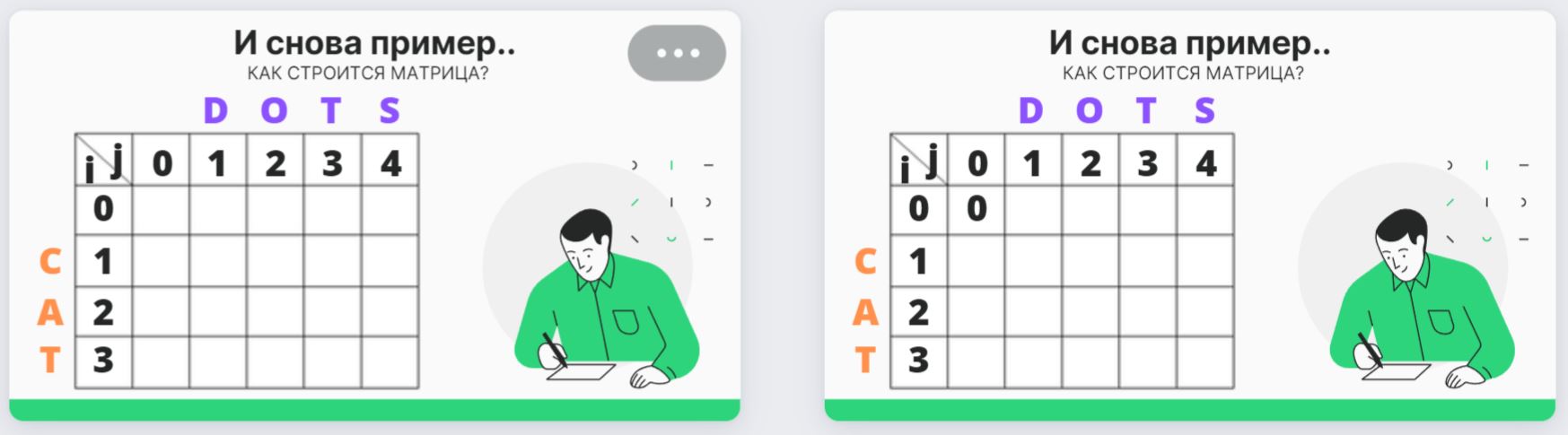
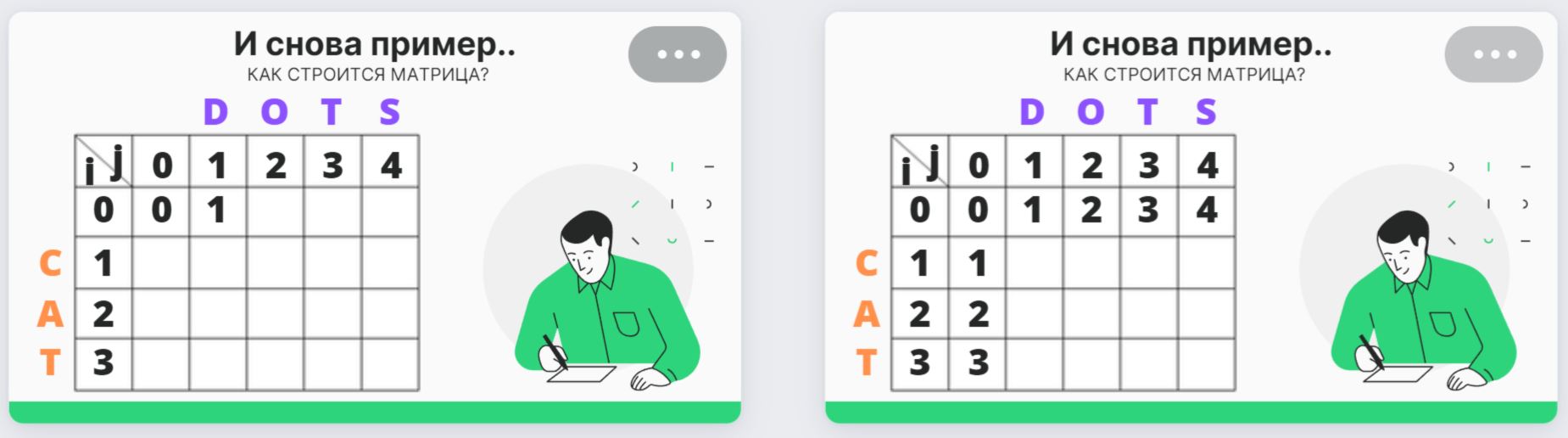
Как расстояние Левенштейна, так и задачу для произвольных цен операций, решает алгоритм *Вагнера — Фишера*, который мы рассмотрим далее.

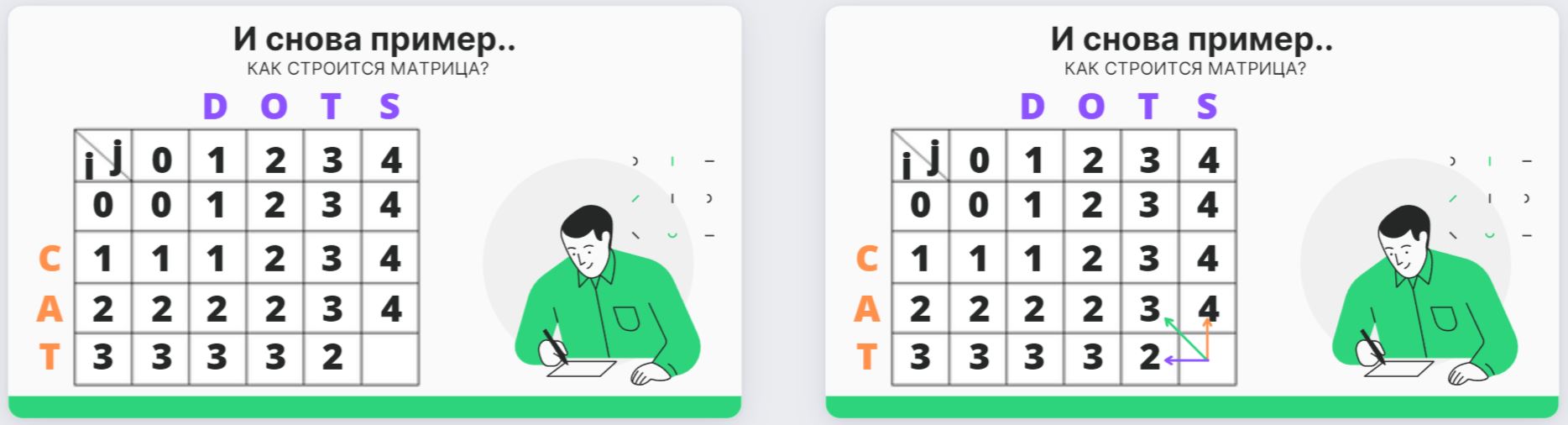
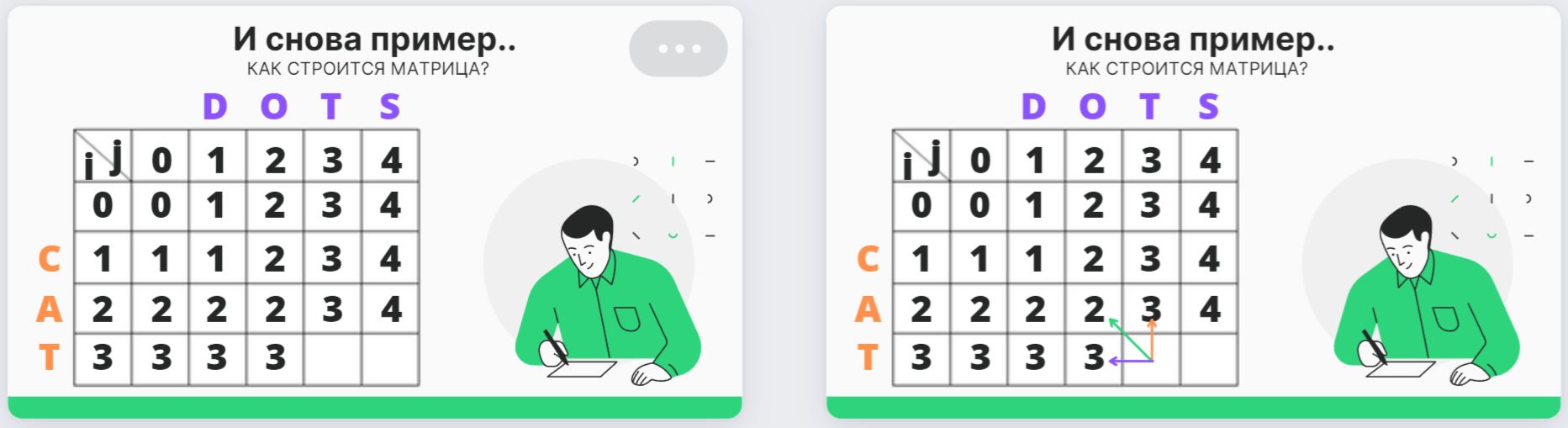
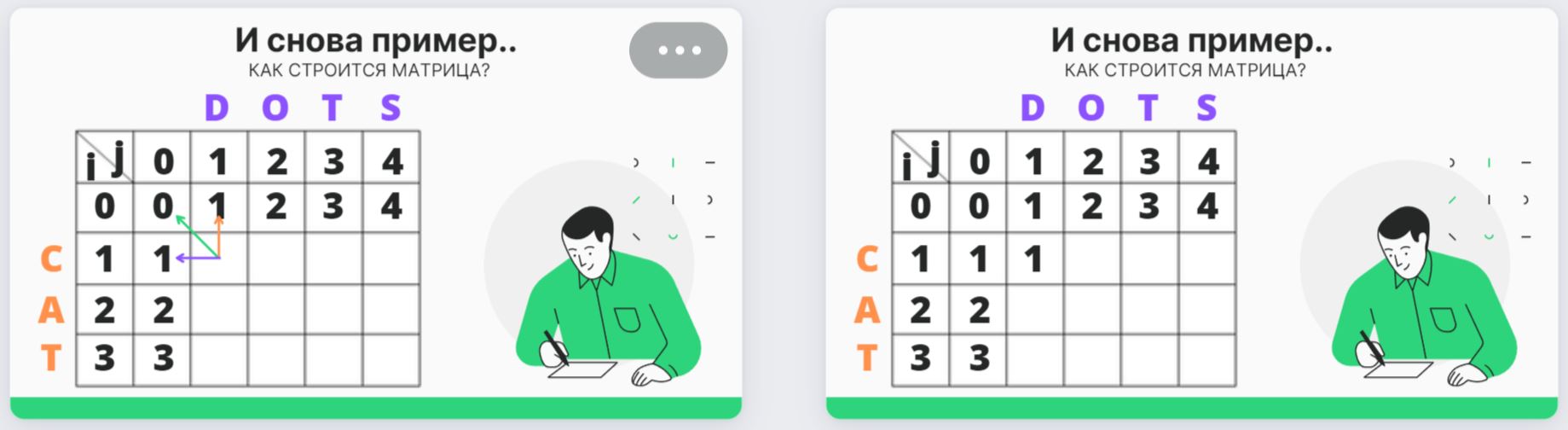
Алгоритм Вагнера – Фишера (1974 г.)

Пусть S1 и S2 – две строки (длиной M и N соответственно) над некоторым алфавитом, тогда расстояние Левенштейна можно подсчитать по следующей рекуррентной формуле:

, где

, где m(a, b) равно нулю, если a = b и единице в противном случае. Результат функции D(i, j) записывается в соответствующую ячейку матрицы размером M\*N.

Рассмотрим на примере, как происходит заполнение матрицы. 



Здесь шаг по i символизирует удаление символа из первой строки, по j — вставку символа в первую строку, а шаг по обоим индексам символизирует замену символа или отсутствие изменений.

Очевидно, справедливы следующие утверждения:

* 
* 
* 

**Дополнительная информация**

*Как вычислить* ***редакционное предписание****?*

*Для восстановления 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) + цена удаления символа S1[i]), добавляем удаление символа S1[i] и идём в (i – 1, j)*
* *если минимально D((i, j – 1) + цена вставки символа S2[j]), добавляем вставку символа S2[j] и идём в (i, j – 1)*
* *если минимально D(i – 1, j – 1) + цена замены символа S1[i] на символ S2[j]), добавляем замену S1[i] на S2[j] (если они не равны; иначе ничего не добавляем), после чего идём в (i – 1, j – 1)*

*Здесь (i, j) — клетка матрицы, в которой мы находимся на данном шаге. Если минимальны два из трёх значений (или равны все три), это означает, что есть 2 или 3 равноценных редакционных предписания.*

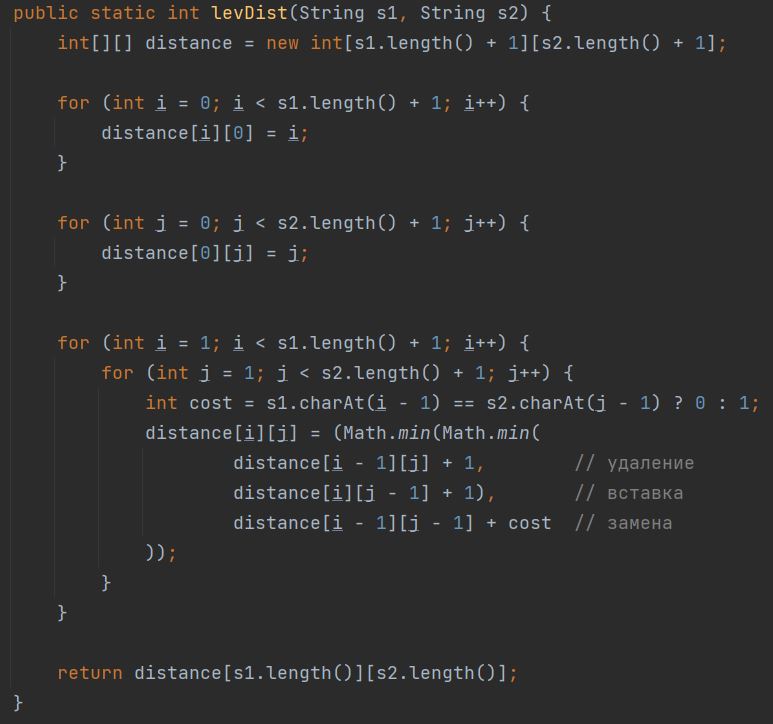
Ключом к эффективности алгоритма является разбиение большой задачи на множество маленьких (динамическое программирование). Например, если известно, что расстояние Левенштейна между последовательностями «cat» и «hat» равно 1, следовательно, расстояние Левенштейна между «cats» и «hat» будет равно 2 (мы воспользовались результатом уже решенной подзадачи, чтобы найти решение более крупной задачи).

**Оценка временной сложности с обоснованием**

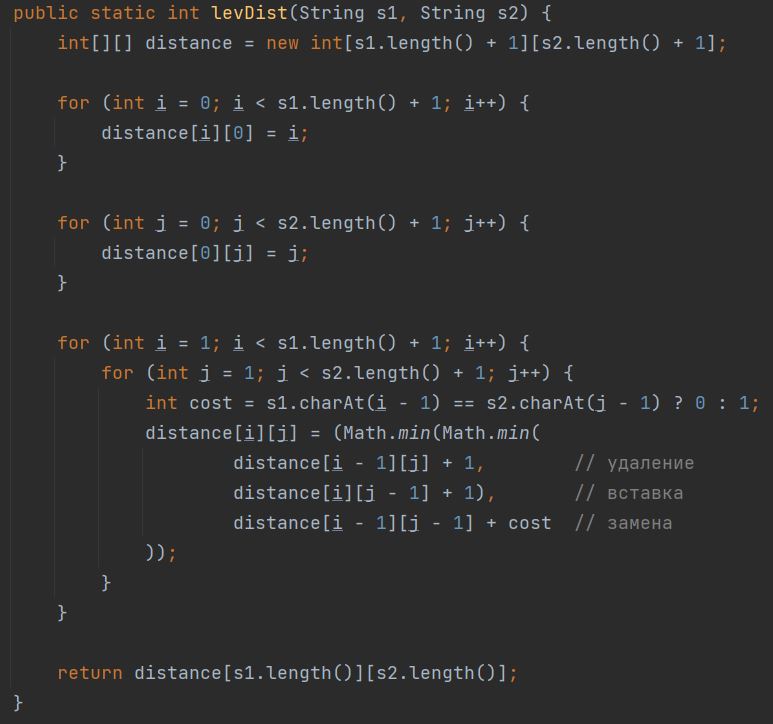
Временная сложность алгоритма – O(M\*N), где M и N – длины последовательностей S1 и S2 соответственно

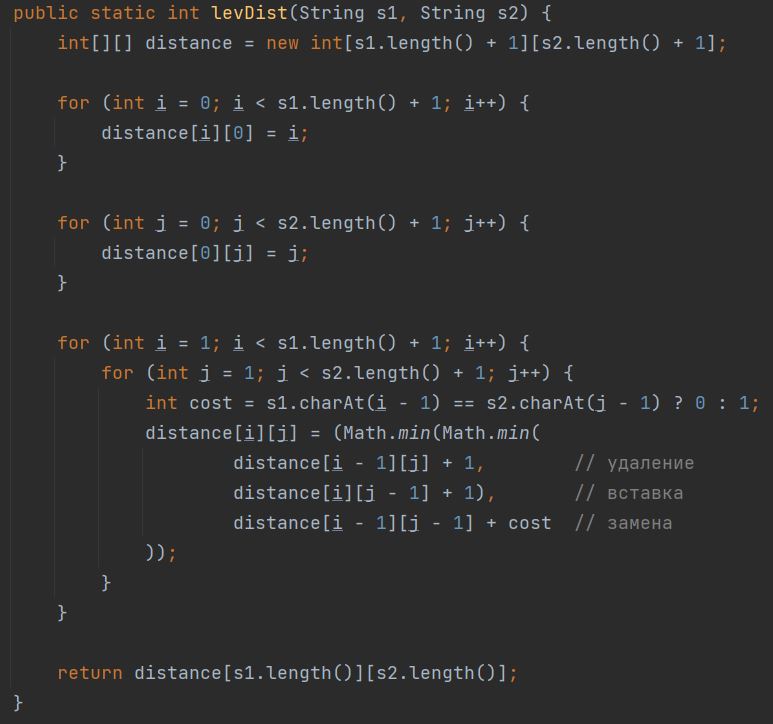
**Доказательство:**

Чтобы вычислить расстояние Левенштейна с помощью алгоритма Вагнера – Фишера, строится матрица размерами M\*N, где M – длина последовательности S1, N – длина последовательности S2.



Затем заполняются первая строка

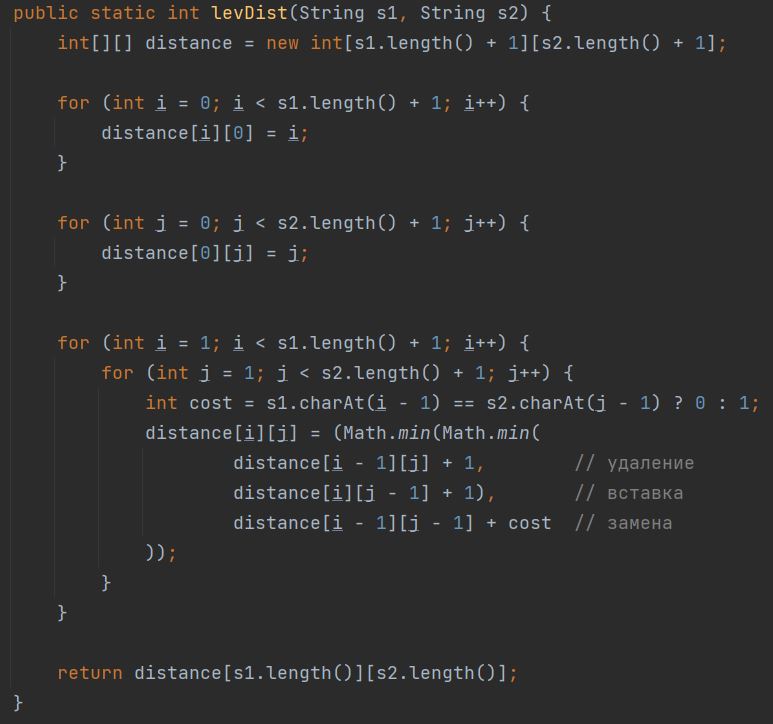


и первый столбец матрицы. 

Временная сложность первого цикла – O(M), т.к. вставка в массив выполняется за константное время, а всего таких вставок – M (т.к. длина первой последовательности S1 равна M символов).

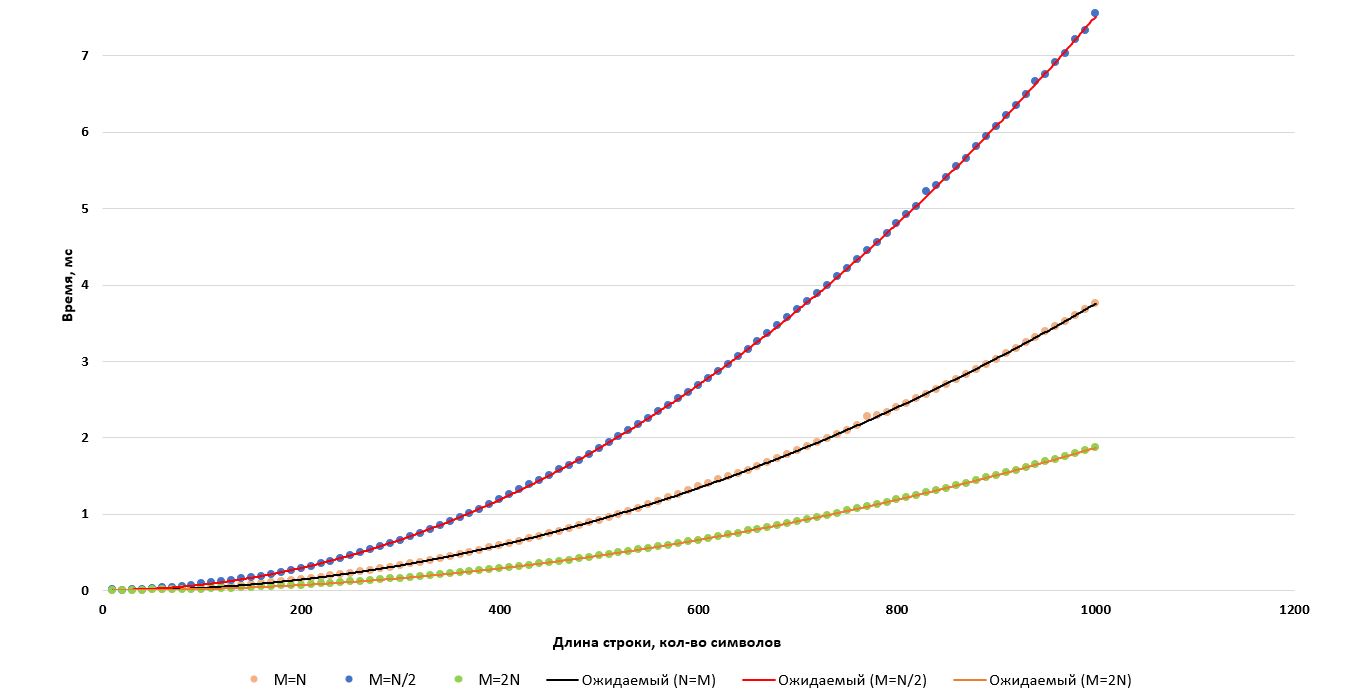
Аналогичные рассуждения производим со вторым циклом. Его сложность – O(N).

Далее происходит заполнение ячеек матрицы:



Внешний цикл пробегается по ячейкам матрицы M раз, внутренний – N раз. Следовательно, всего цикл пройдется по массиву M\*N раз. И так как операция вставки элемента в массив имеет константную сложность, то временная сложность вложенного цикла – O(M\*N).

Таким образом, получаем, что временная сложность алгоритма вычисления расстояния Левенштейна равна: O(M) + O(N) + O(M\*N) = O(M\*N). Что и требовалось доказать.

Графики

Как видим, полученные графики не отличается от ожидаемых в теории.

**4. Выводы. Плюсы и минусы алгоритма, его применимость**

Применимость:

Расстояние Левенштейна (и его обобщения) активно применяется:

* Для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи)
* Для сравнения текстовых файлов
* В биоинформатике для сравнения генов, хромосом и белков
* Для проверки текстов на плагиат
* В лингвистике для определения насколько языки или диалекты отличаются друг от друга

Минусы:

* При перестановке местами слов или частей слов получаются сравнительно большие расстояния
* Расстояния между совершенно разными короткими словами оказываются небольшими, в то время как расстояния между очень похожими длинными словами оказываются значительными

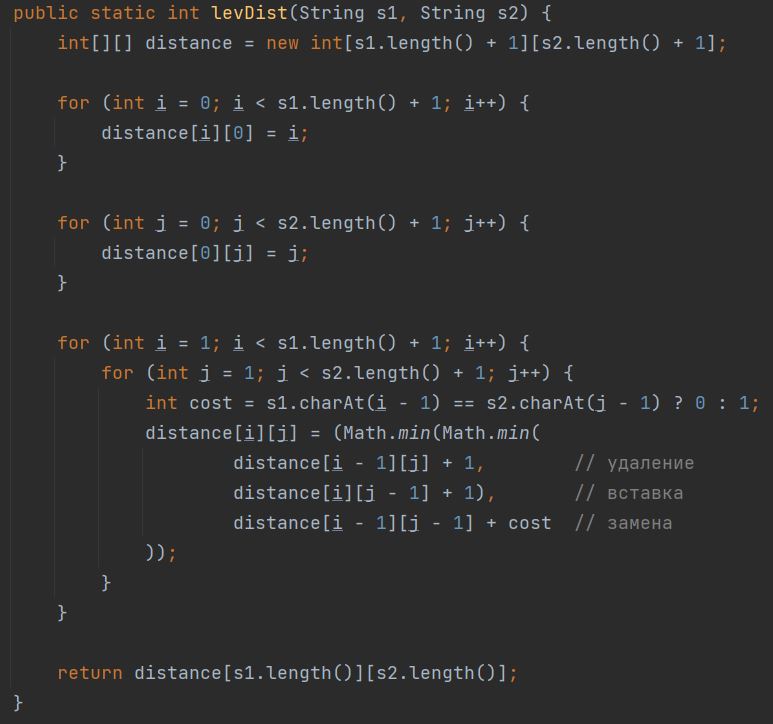
Плюсы:

* Алгоритм прост для понимания и в реализации
* Алгоритм имеет большое количество применений
* Если нам требуется найти только расстояние (без редакционного предписания), то алгоритм можно оптимизировать и получить сложность по памяти – O(min(M, N)). Примечание: *для этого учитывается то, что после вычисления любой строки матрицы предыдущая строка больше не нужна*

**5. Список использованной литературы (источников)**

1. <https://ru.wikipedia.org/wiki/Расстояние_Левенштейна#cite_note-1> (поменять на англ)
2. <https://www.baeldung.com/cs/levenshtein-distance-computation>

**6. Код программы**

****

**7. Ссылка на репозиторий**

[**https://github.com/bekmnsrw/aisd-spring-2022**](https://github.com/bekmnsrw/aisd-spring-2022)

(код, тесты, таблица, графики, презентация и отчет в репозитории)