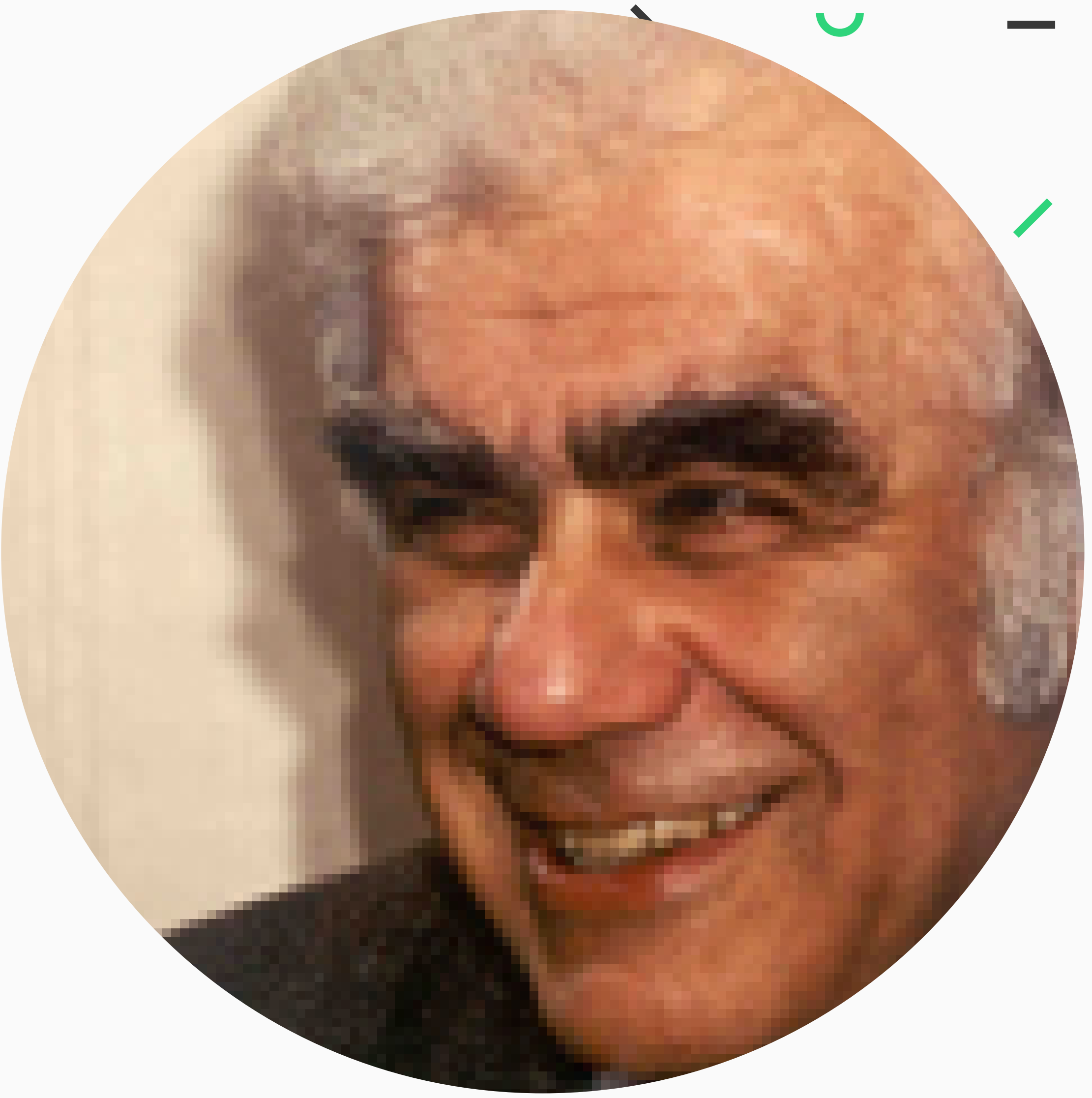
Вычисление расстояния Левенштейна

БЕКМАНСУРОВ ИЛЬЯ ГР. 11-102

Историческая справка

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

Впервые задачу поставил в 1965 г. советский математик Владимир Левенштейн



Принцип устройства

Пусть имеются две строки S1 и S2. Мы хотим перевести одну в другую, используя следующие операции:

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

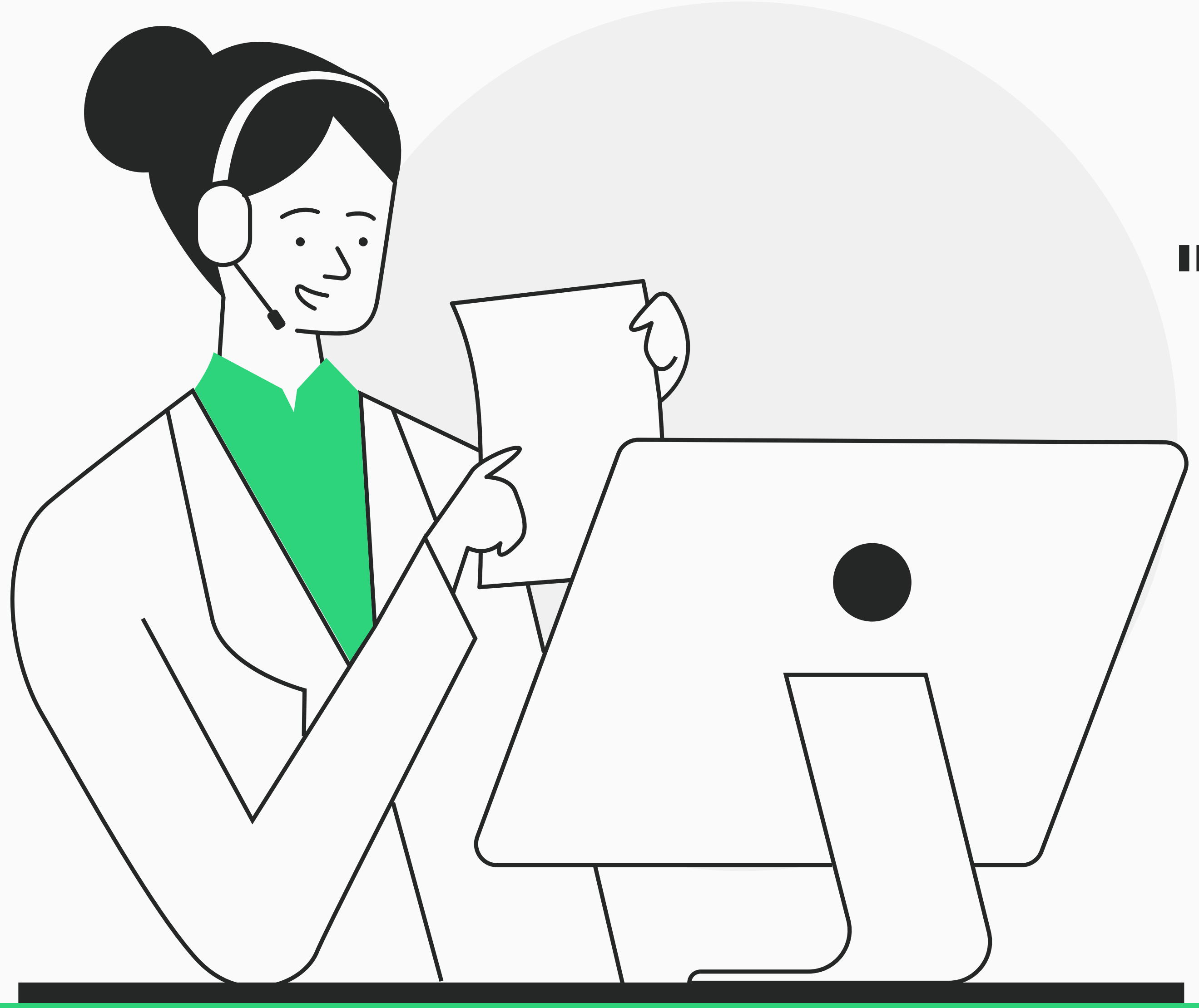
Тогда минимальное количество вышеперечисленных операций для перевода S1 в S2 называется **редакционным расстоянием**, а их последовательность - **редакционным предписанием**

Рассмотрим пример

"СКРИПКАЛ ИСА" == "СКРИП КОЛЕСА" ?

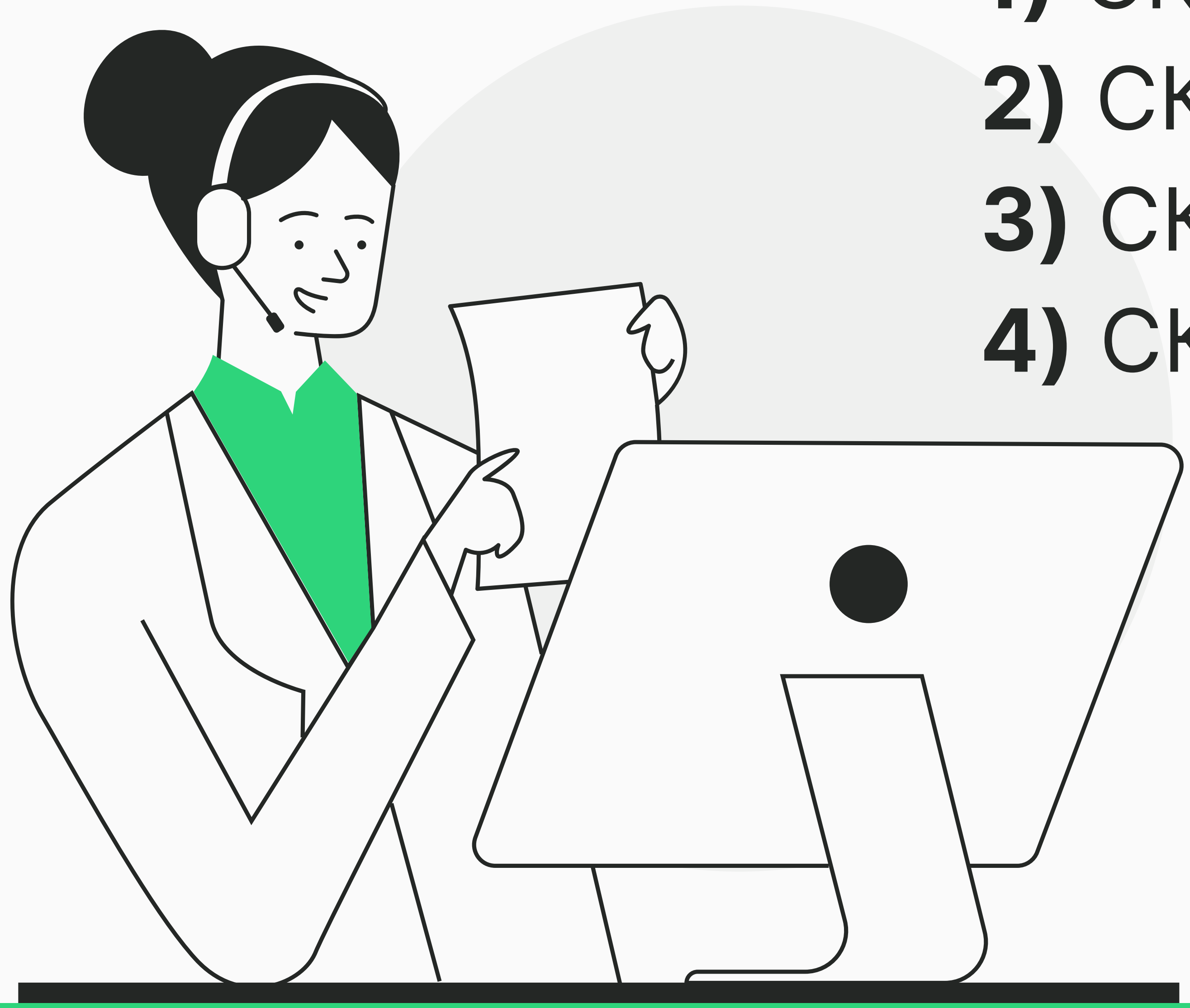
или

"СКРИПКАЛ ИСА" == "СКРИПКА ЛИСА" ?



Рассмотрим пример

"СКРИПКАЛ ИСА" = "СКРИП КОЛЕСА" ?

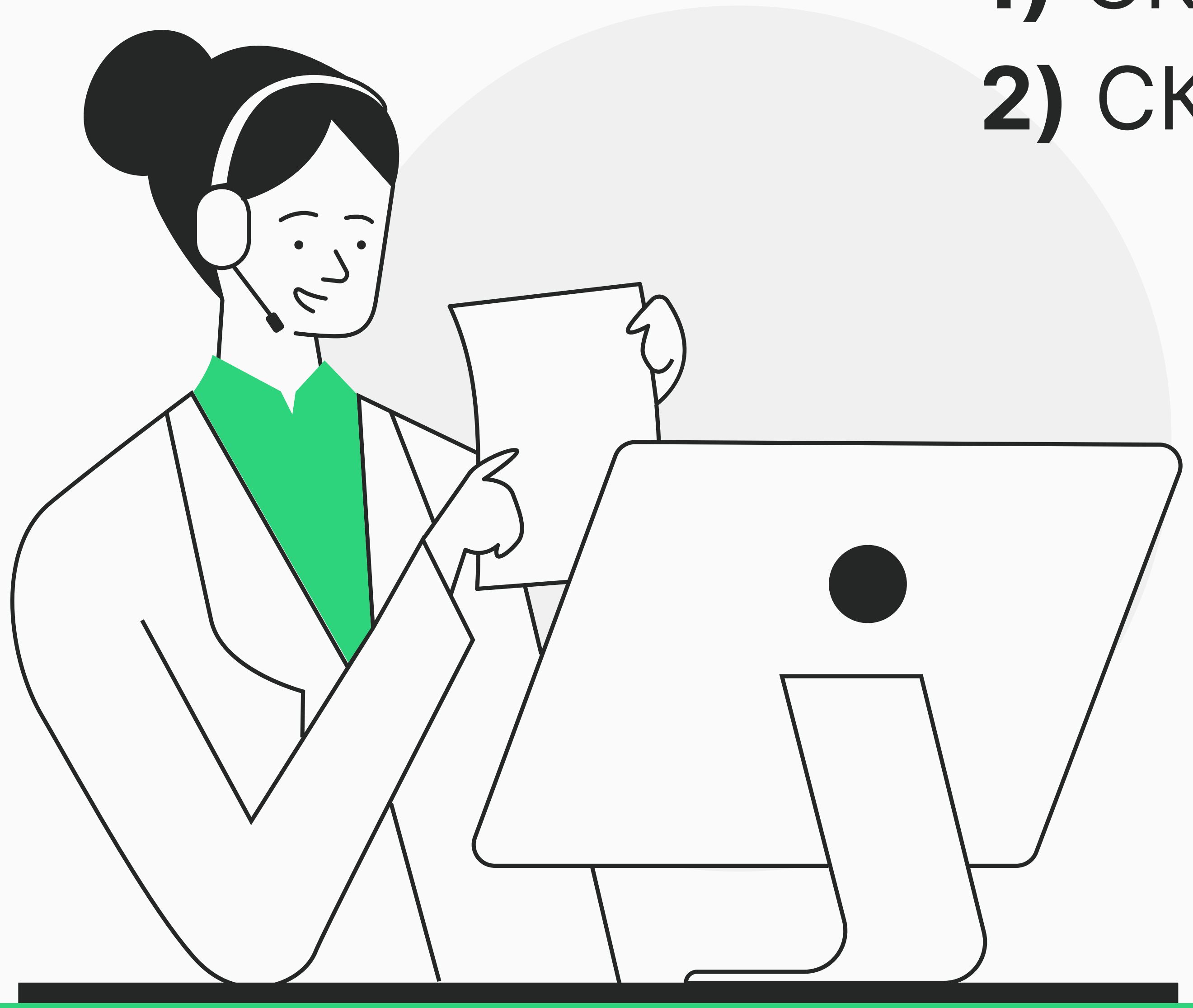


1) СКРИПКАЛ ИСА -> СКРИП КАЛ ИСА
2) СКРИП КАЛ ИСА -> СКРИП КОЛ ИСА
3) СКРИП КОЛ ИСА -> СКРИП КОЛИСА
4) СКРИП КОЛИСА -> СКРИП КОЛЕСА

Рассмотрим пример

"СКРИПКАЛ ИСА" = "СКРИПКА ЛИСА" ?

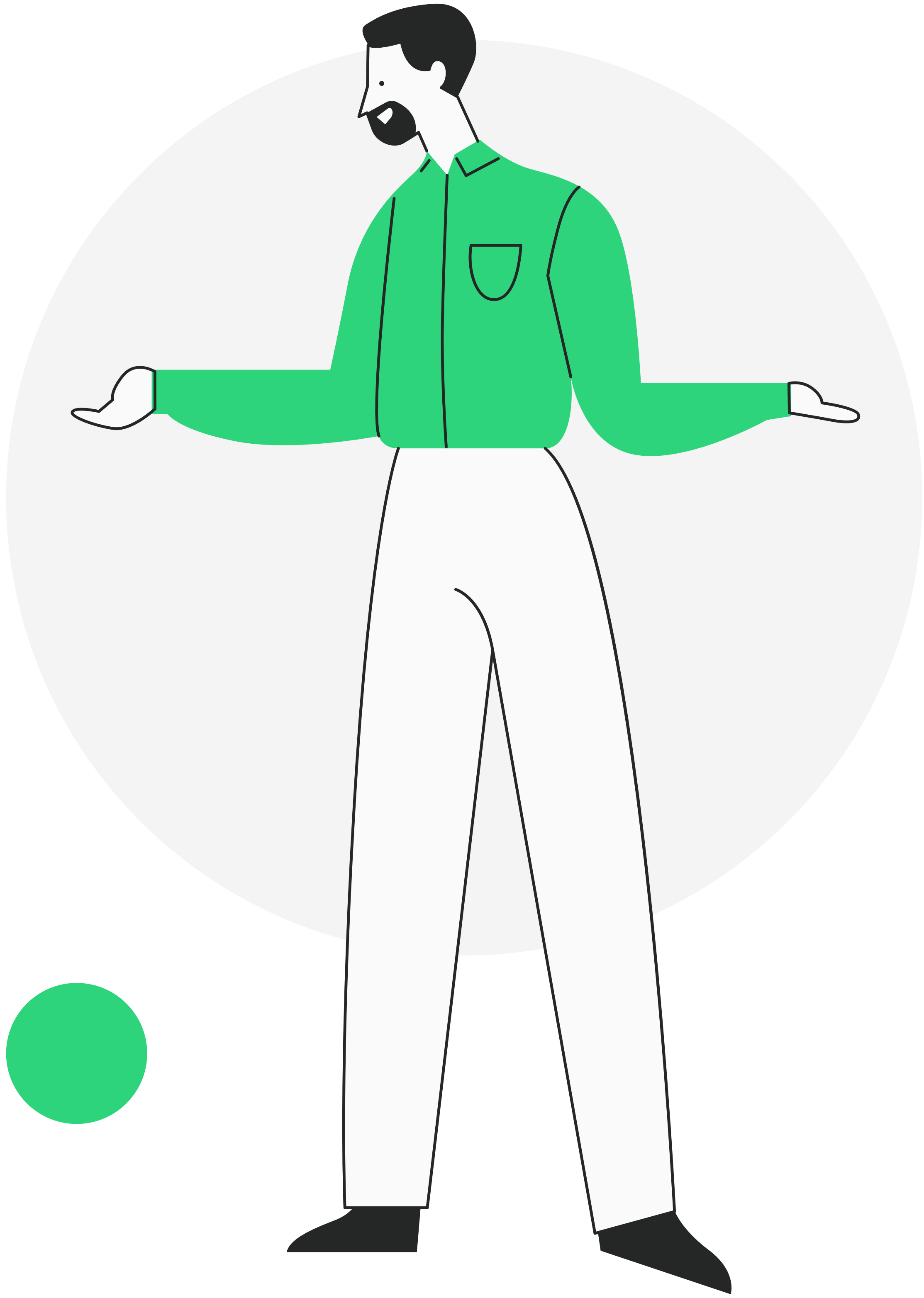
1) СКРИПКАЛ ИСА -> СКРИПКА Л ИСА
2) СКРИПКА Л ИСА -> СКРИПКА ЛИСА



Стоимости операций

Для нахождения редакционного расстояния цены операций удаления, вставки и замены символа 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, a)** = 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** (1974), ОСНОВАННЫЙ НА **ДИНАМИЧЕСКОМ ПРОГРАММИРОВАНИИ**

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

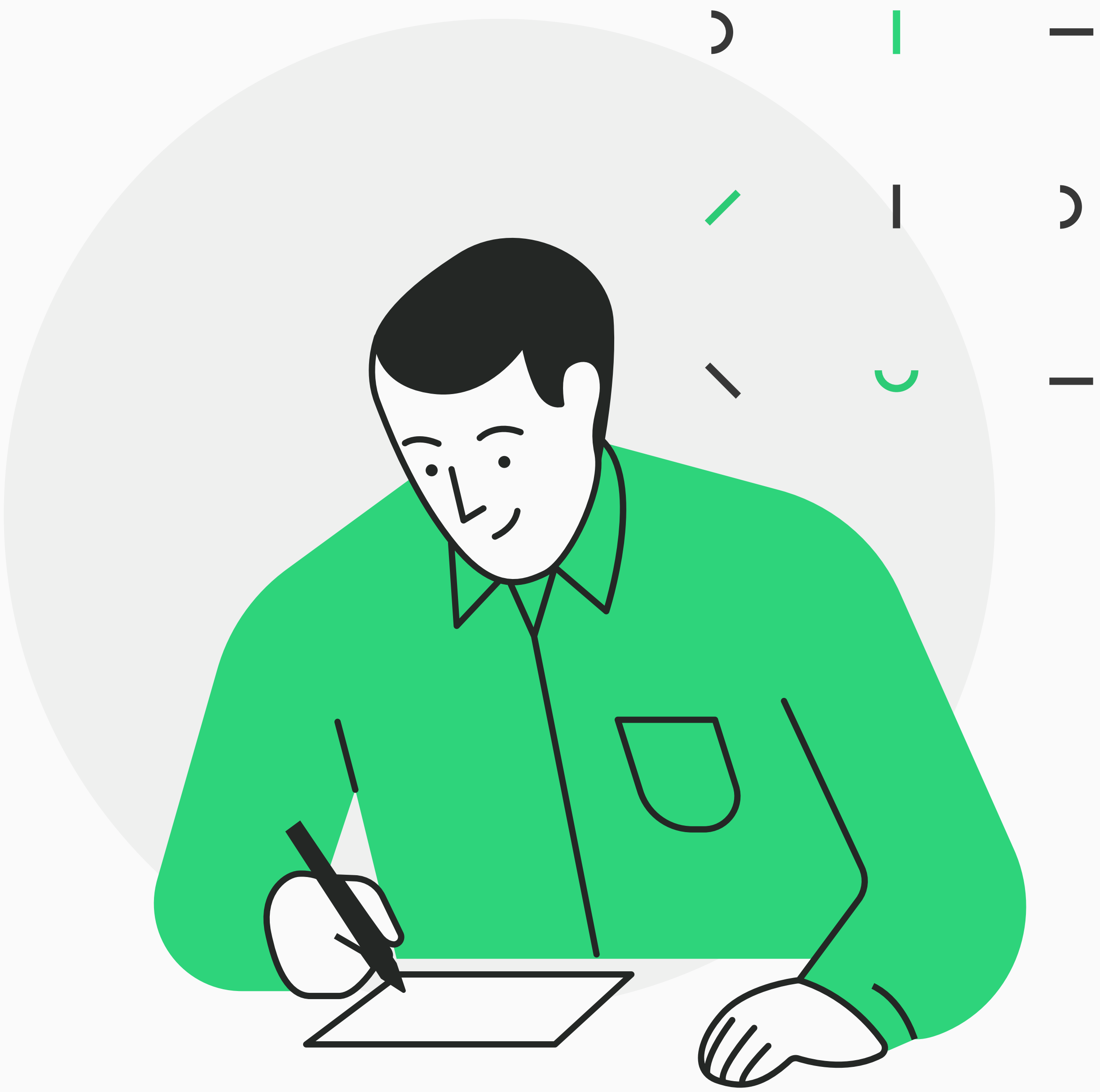
$$D(i,j) = \begin{cases} 0, & i=0,\ j=0 \\ i, & j=0,\ i>0 \\ j, & i=0,\ j>0 \\ \min\{ & \\ \qquad D(i,j-1)+1, & \\ \qquad D(i-1,j)+1, & j>0,\ i>0 \\ \qquad D(i-1,j-1)+\mathrm{m}(S_1[i],S_2[j]) & \\ \} & \end{cases}$$

где m(a, b) = 0, если a = b и единице в противном случае. Результат функции D(i, j) записывается в соответствующую ячейку матрицы размером M*N

И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

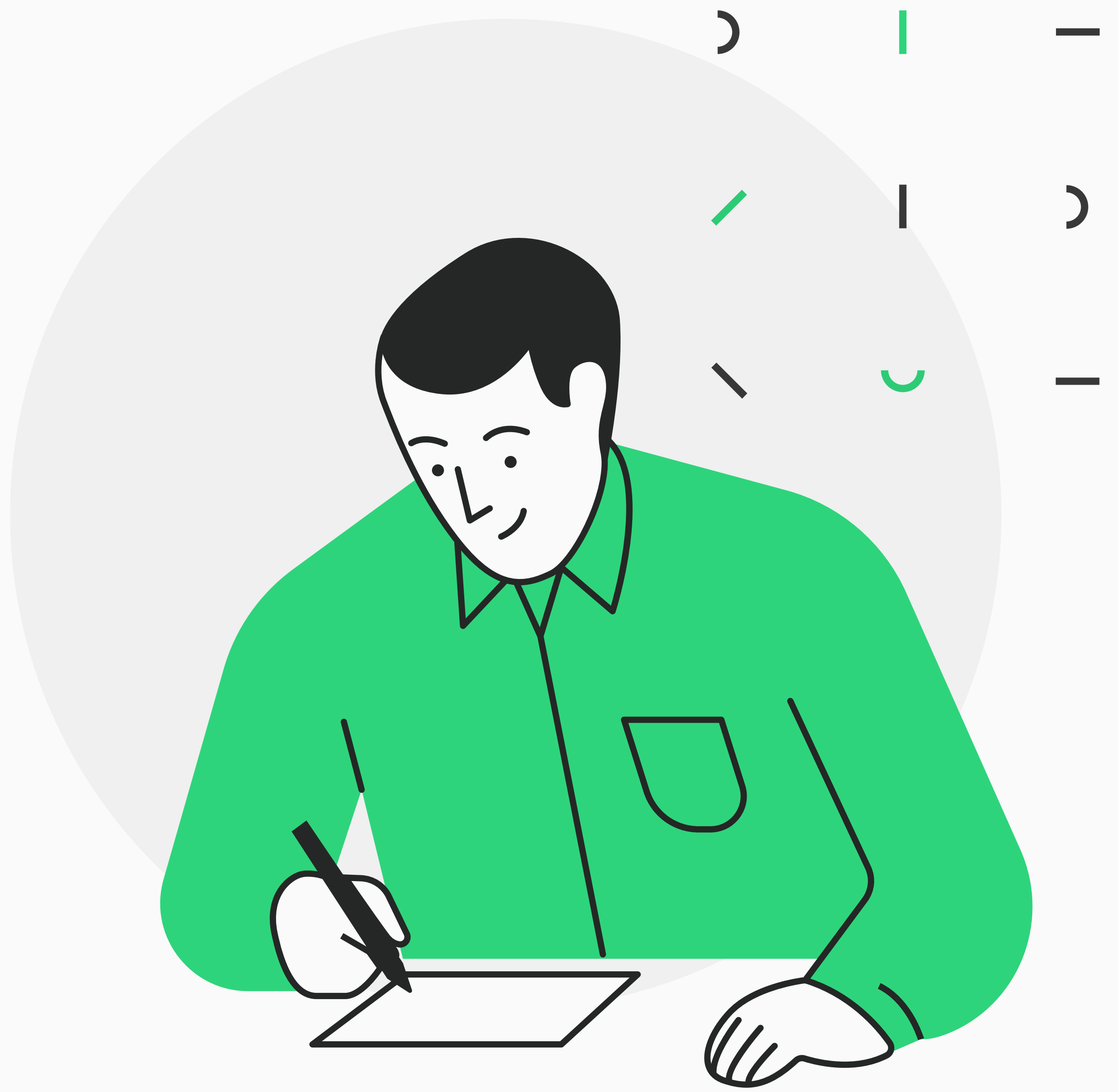
			D	O	T	S
	i \ j	0	1	2	3	4
	0					
C	1					
A	2					
T	3					



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

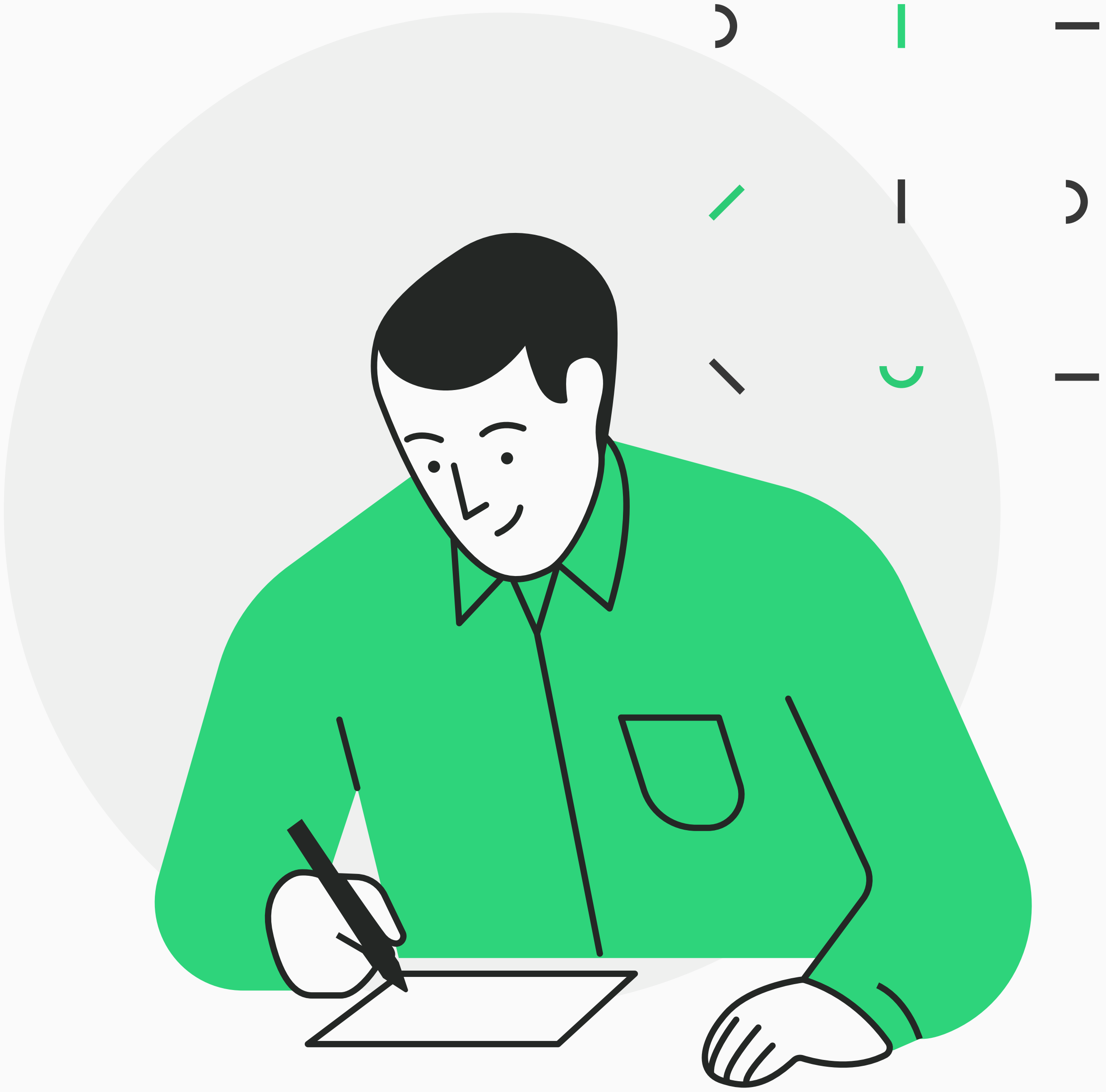
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0				
C	1					
A	2					
T	3					



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

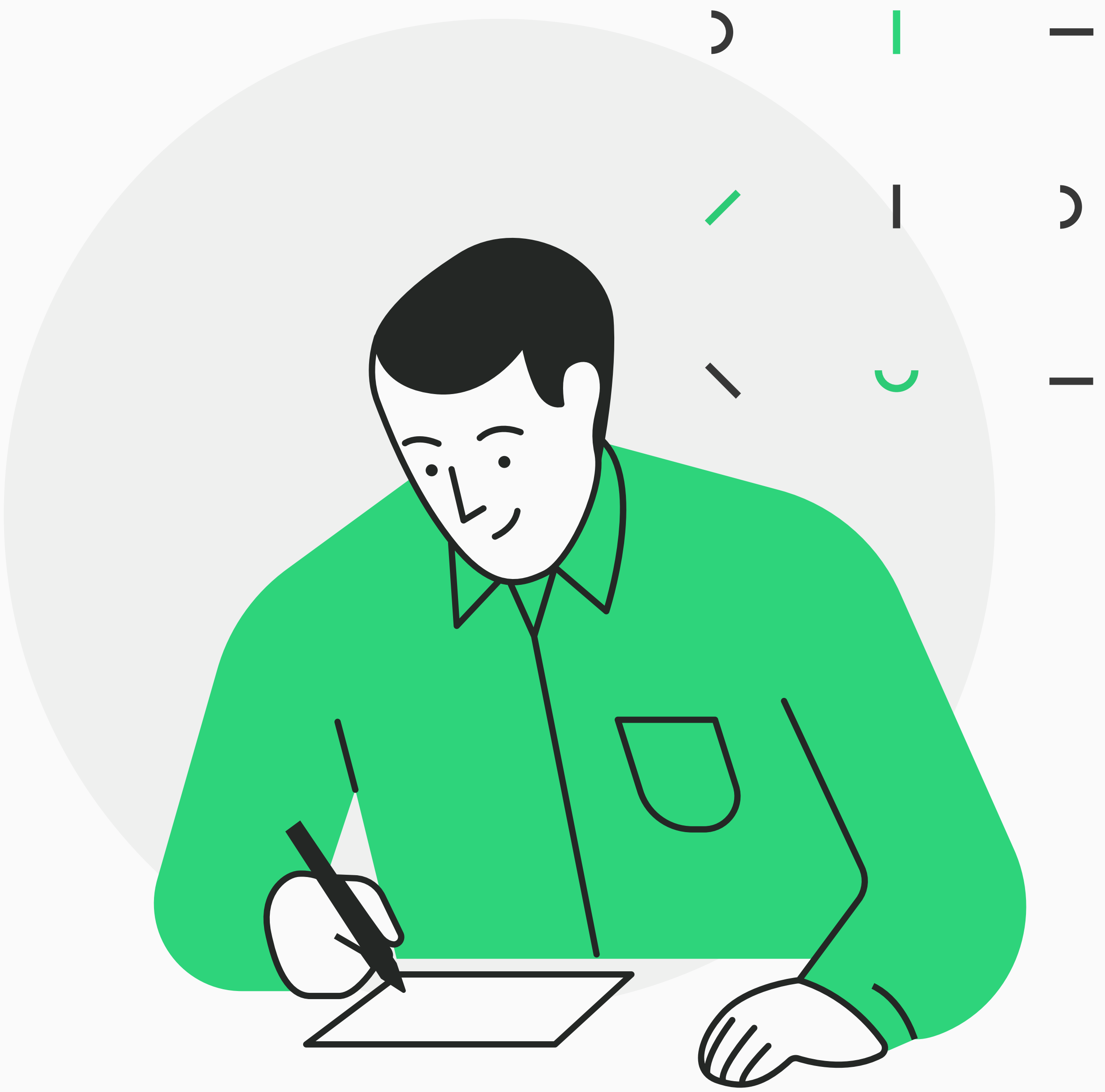
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1			
C	1					
A	2					
T	3					



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

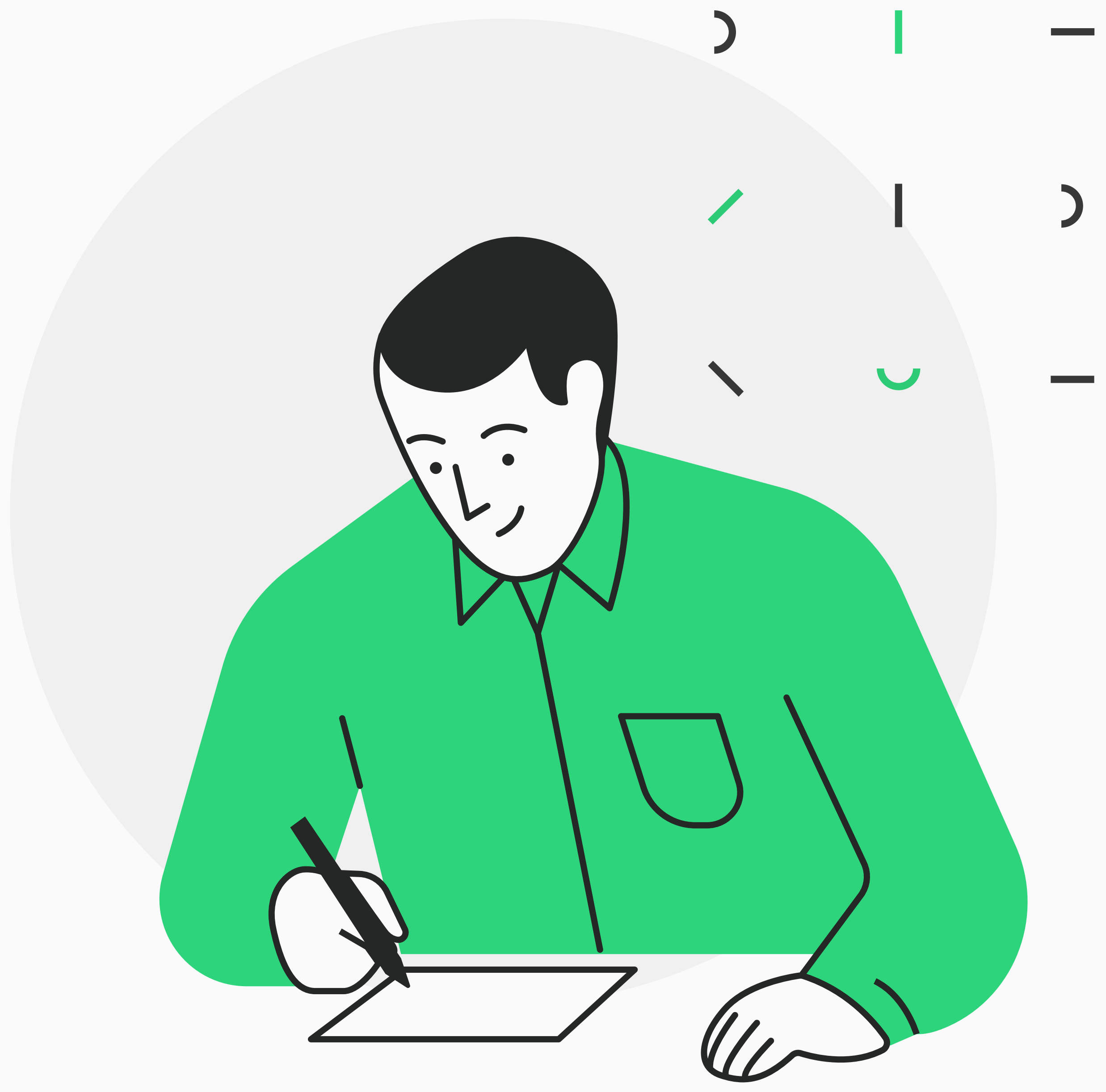
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1				
A	2	2				
T	3	3				



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

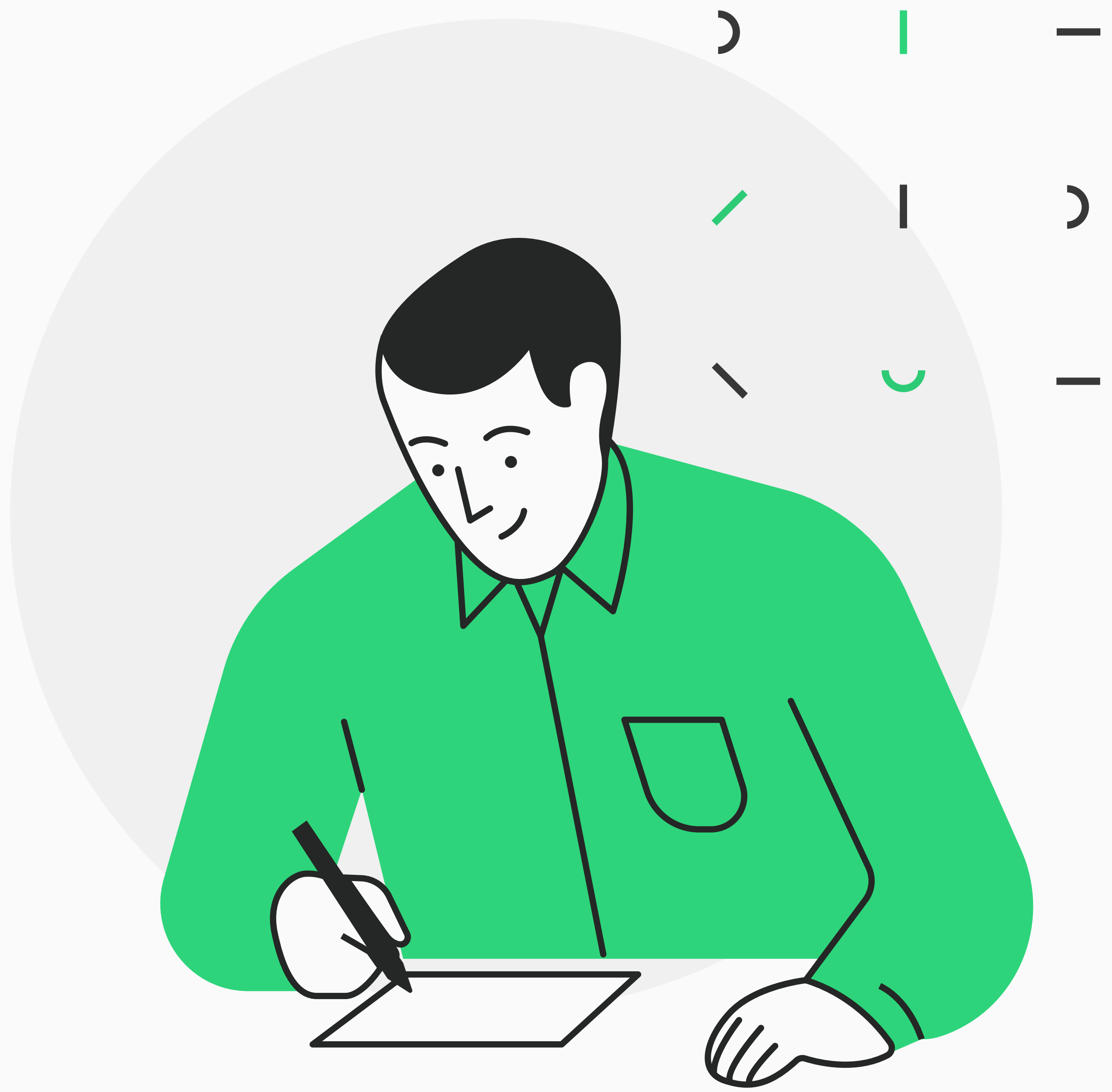
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1				
A	2	2				
T	3	3				



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

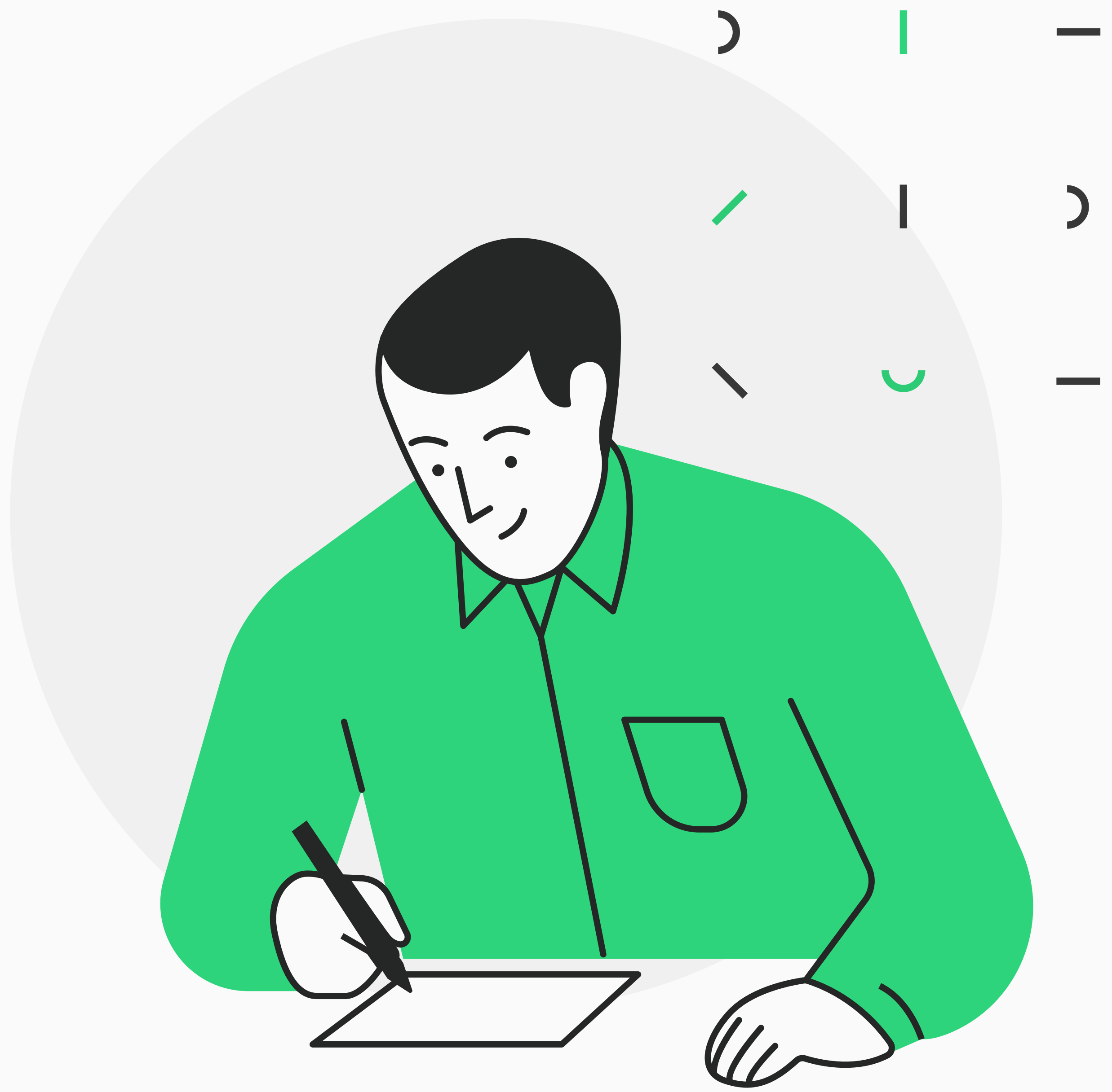
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1	1			
A	2	2				
T	3	3				



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

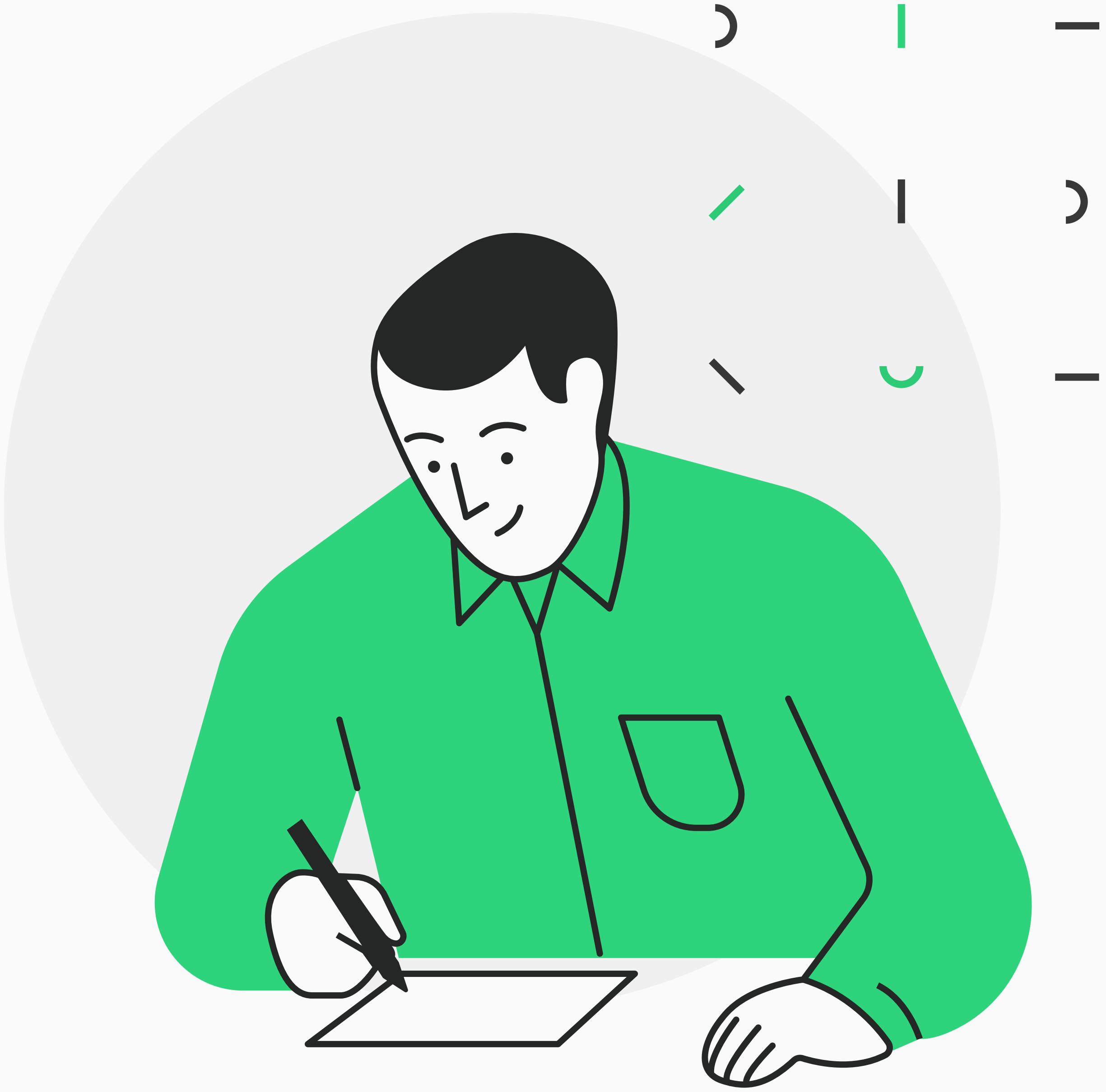
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1	1	2	3	4
A	2	2	2	2	3	4
T	3	3	3	3		



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

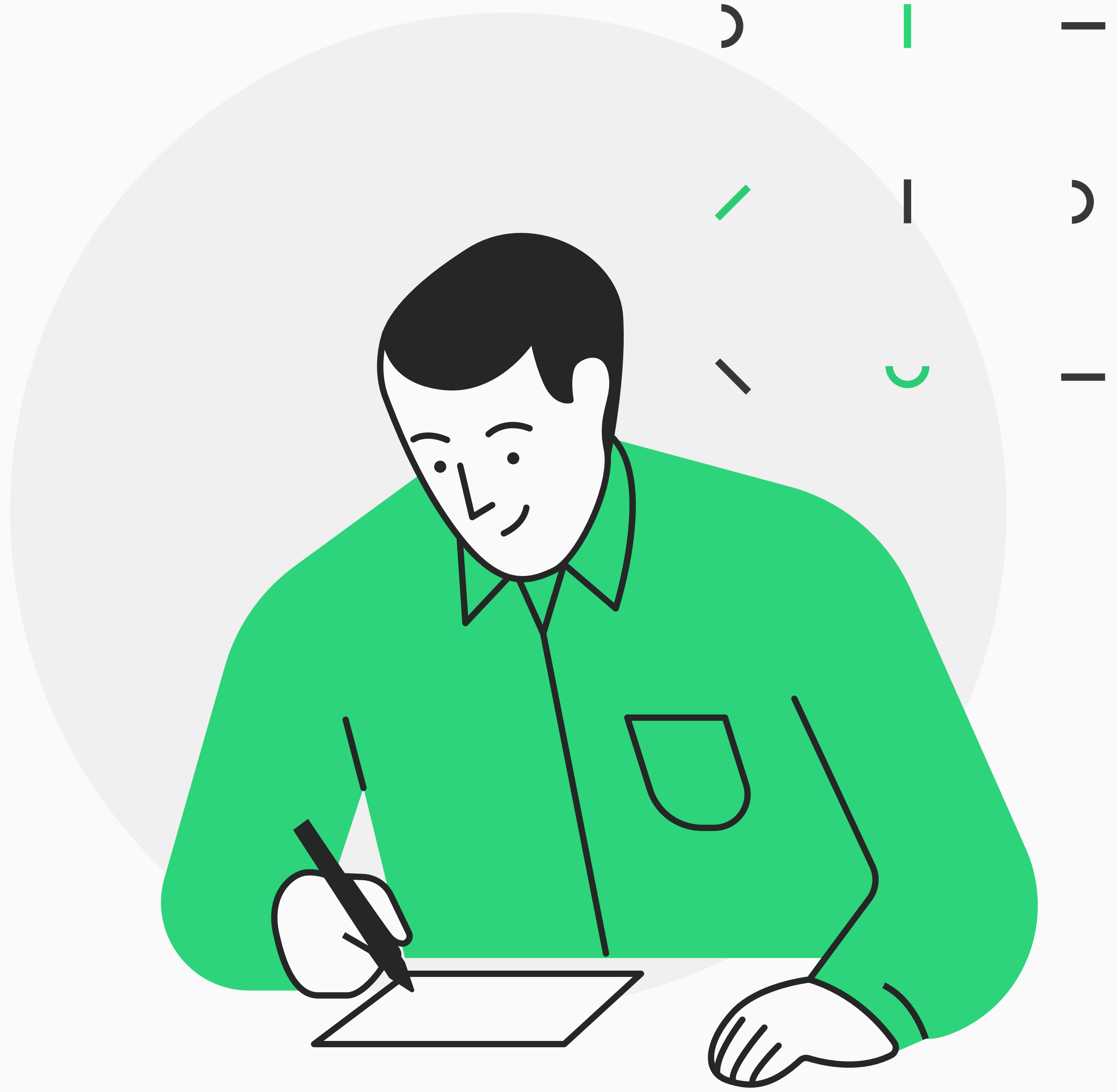
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1	1	2	3	4
A	2	2	2	2	3	4
T	3	3	3	3	3	



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

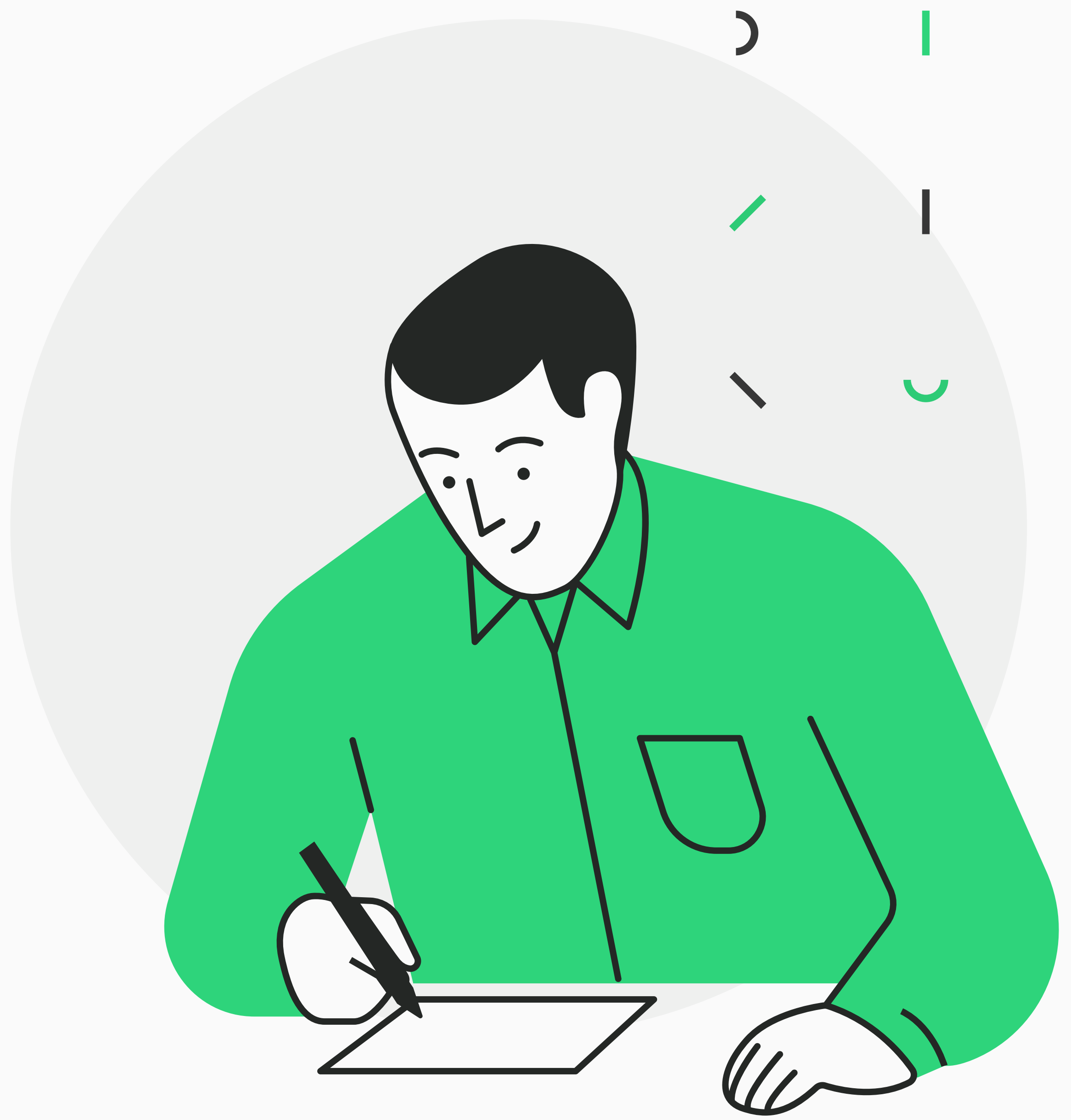
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1	1	2	3	4
A	2	2	2	2	3	4
T	3	3	3	3	2	



И снова пример..

КАК СТРОИТСЯ МАТРИЦА?

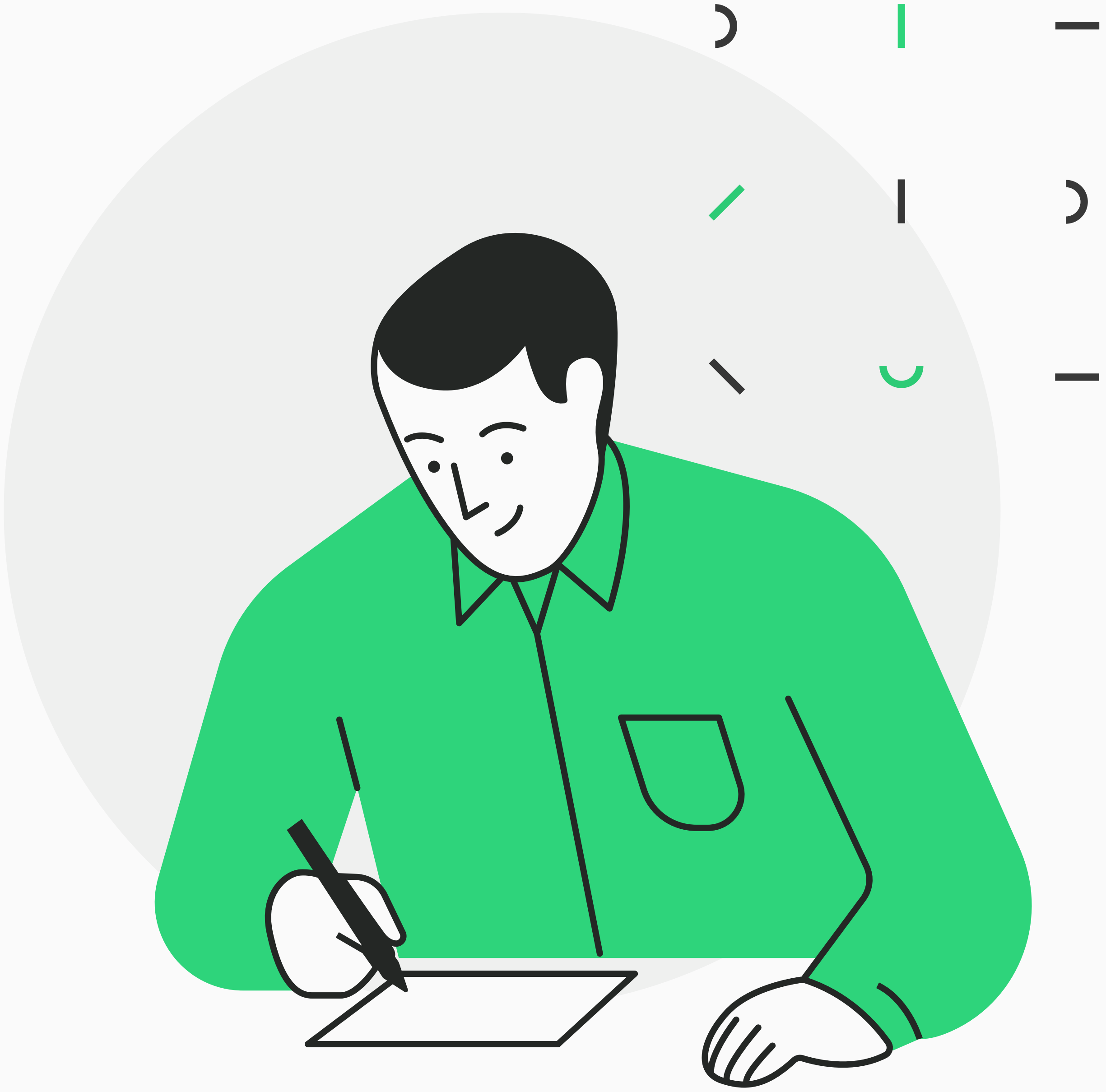
			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1	1	2	3	4
A	2	2	2	2	3	4
T	3	3	3	3	2	



И снова пример..

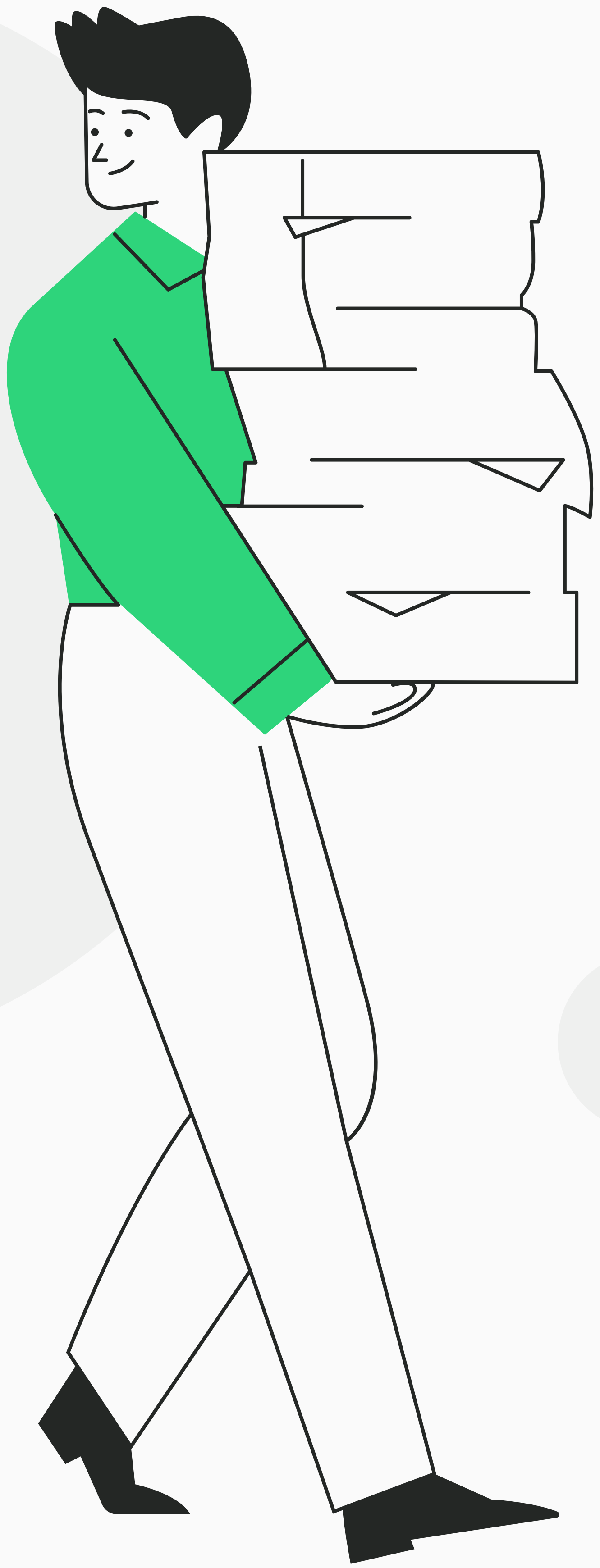
КАК СТРОИТСЯ МАТРИЦА?

			D	O	T	S
	i \ j	0	1	2	3	4
	0	0	1	2	3	4
C	1	1	1	2	3	4
A	2	2	2	2	3	4
T	3	3	3	3	2	3



```java
public static int levDist(String s1, String s2) {
    int[][] distance = new int[s1.length() + 1][s2.length() + 1];

    for (int i = 0; i < s1.length() + 1; i++) {
        distance[i][0] = i;
    }

    for (int j = 0; j < s2.length() + 1; j++) {
        distance[0][j] = j;
    }

    for (int i = 1; i < s1.length() + 1; i++) {
        for (int j = 1; j < s2.length() + 1; j++) {
            int cost = s1.charAt(i - 1) == s2.charAt(j - 1) ? 0 : 1;
            distance[i][j] = (Math.min(Math.min(
                    distance[i - 1][j] + 1,         // удаление
                    distance[i][j - 1] + 1),        // вставка
                    distance[i - 1][j - 1] + cost  // замена
            ));
        }
    }

    return distance[s1.length()][s2.length()];
}
```

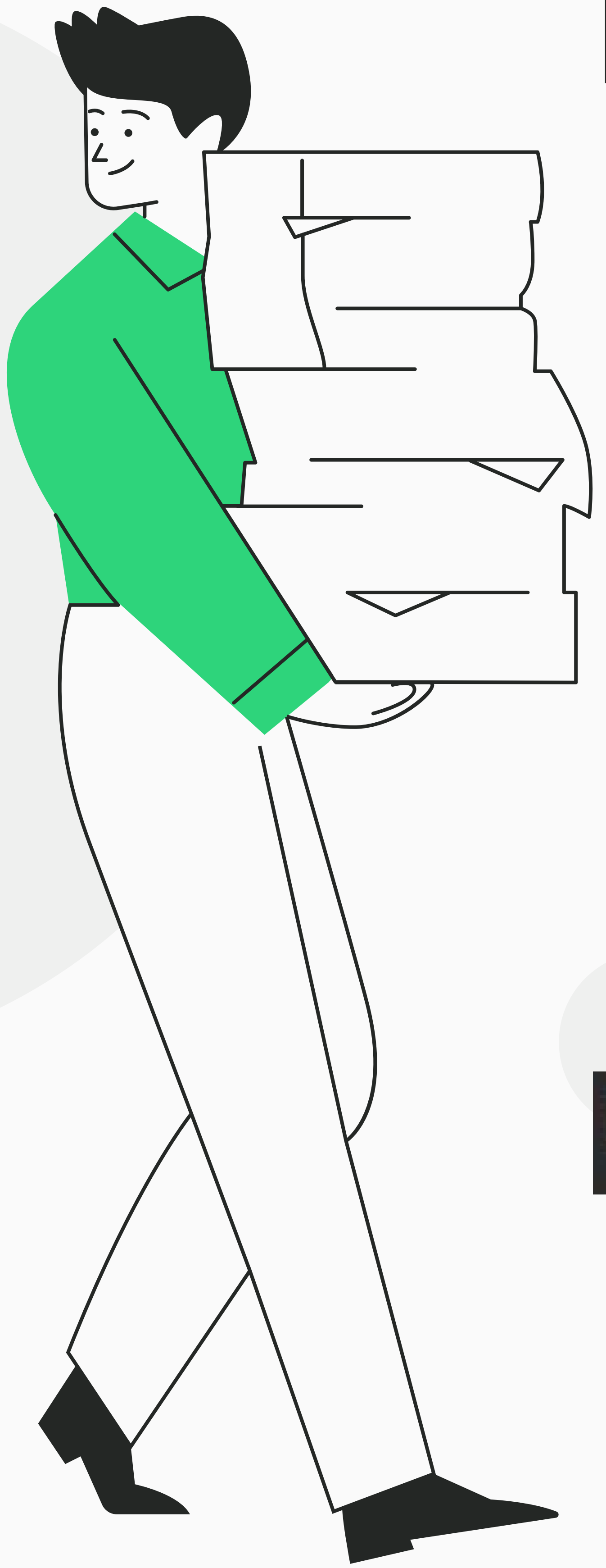
Временная сложность



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

Временная сложность

ДОКАЗАТЕЛЬСТВО

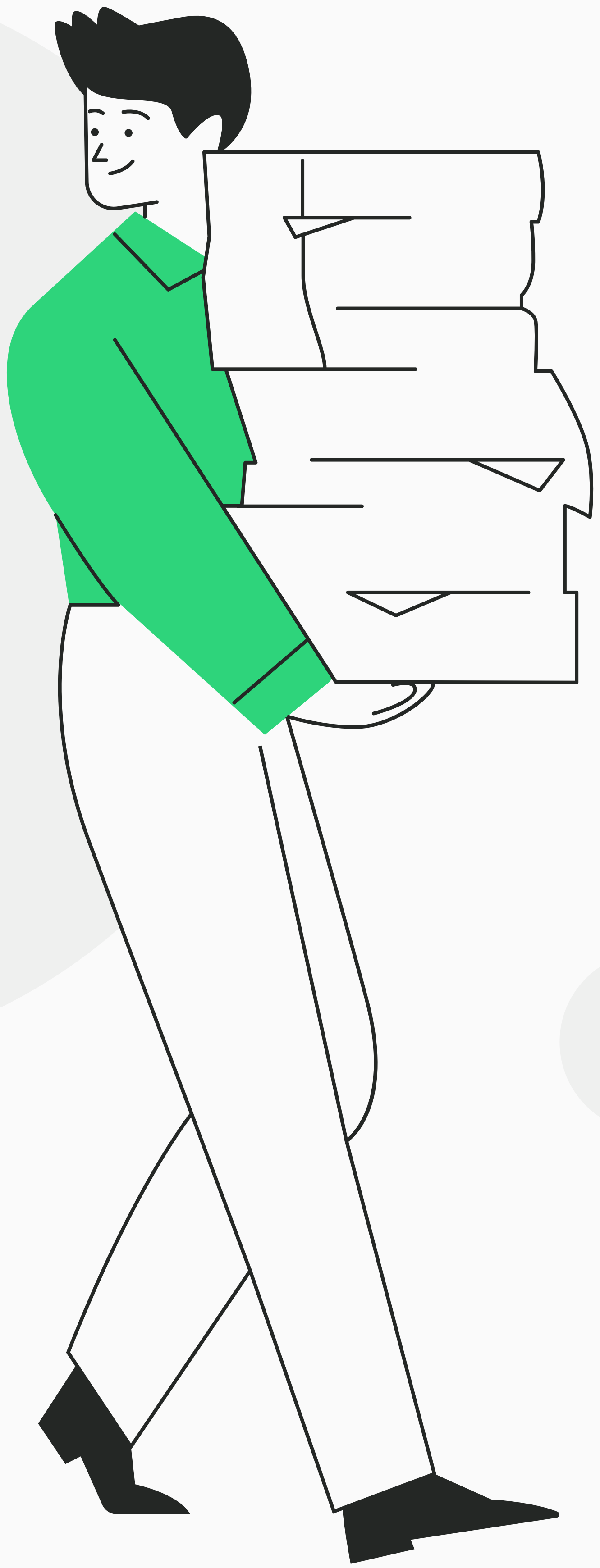


Чтобы вычислить расстояние Левенштейна с помощью алгоритма Вагнера – Фишера, строится матрица размерами M*N, где M – длина последовательности S1, N – длина последовательности S2

```
int[][] distance = new int[s1.length() + 1][s2.length() + 1];
```

Временная сложность

ДОКАЗАТЕЛЬСТВО



Затем заполняются первая строка

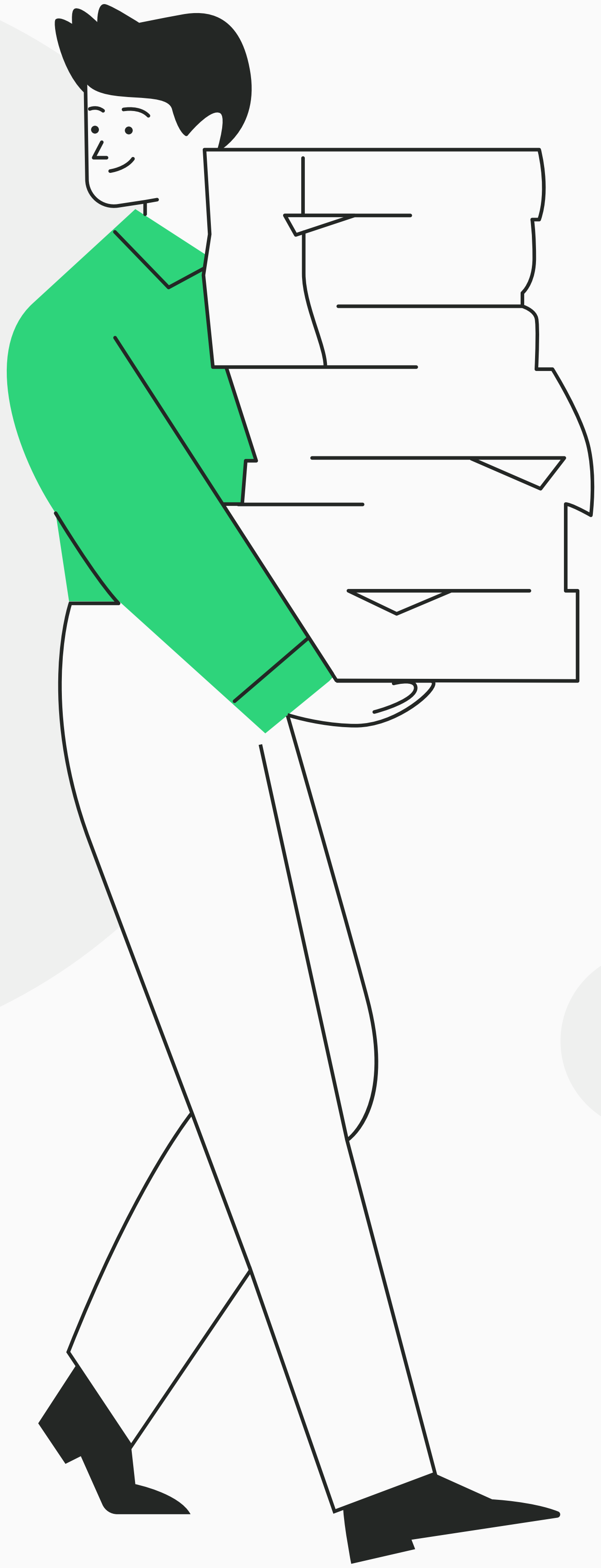
```
for (int i = 0; i < s1.length() + 1; i++) {
    distance[i][0] = i;
}
```

и первый столбец матрицы

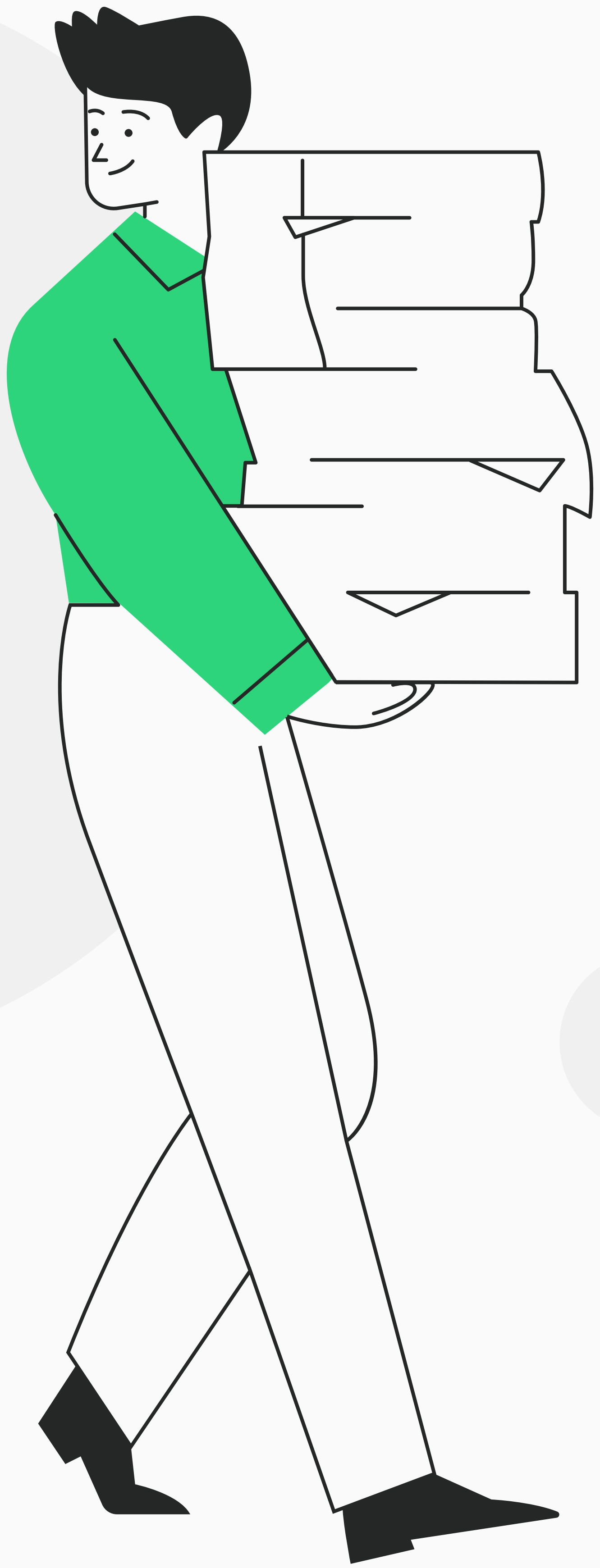
```
for (int j = 0; j < s2.length() + 1; j++) {
    distance[0][j] = j;
}
```

Временная сложность

ДОКАЗАТЕЛЬСТВО



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)

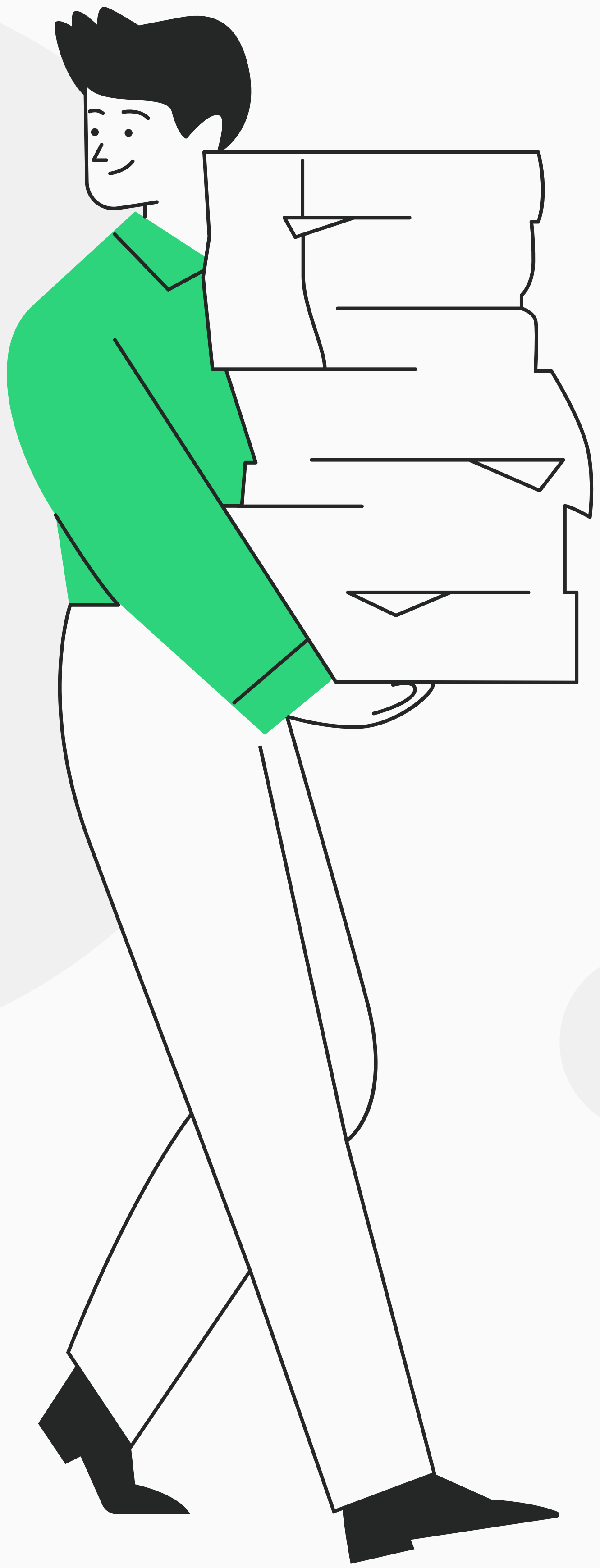


Временная сложность

ДОКАЗАТЕЛЬСТВО

Далее происходит заполнение ячеек матрицы:

```
for (int i = 1; i < s1.length() + 1; i++) {
    for (int j = 1; j < s2.length() + 1; j++) {
        int cost = s1.charAt(i - 1) == s2.charAt(j - 1) ? 0 : 1;
        distance[i][j] = (Math.min(Math.min(
                distance[i - 1][j] + 1,         // удаление
                distance[i][j - 1] + 1),        // вставка
                distance[i - 1][j - 1] + cost  // замена
        ));
    }
}
```



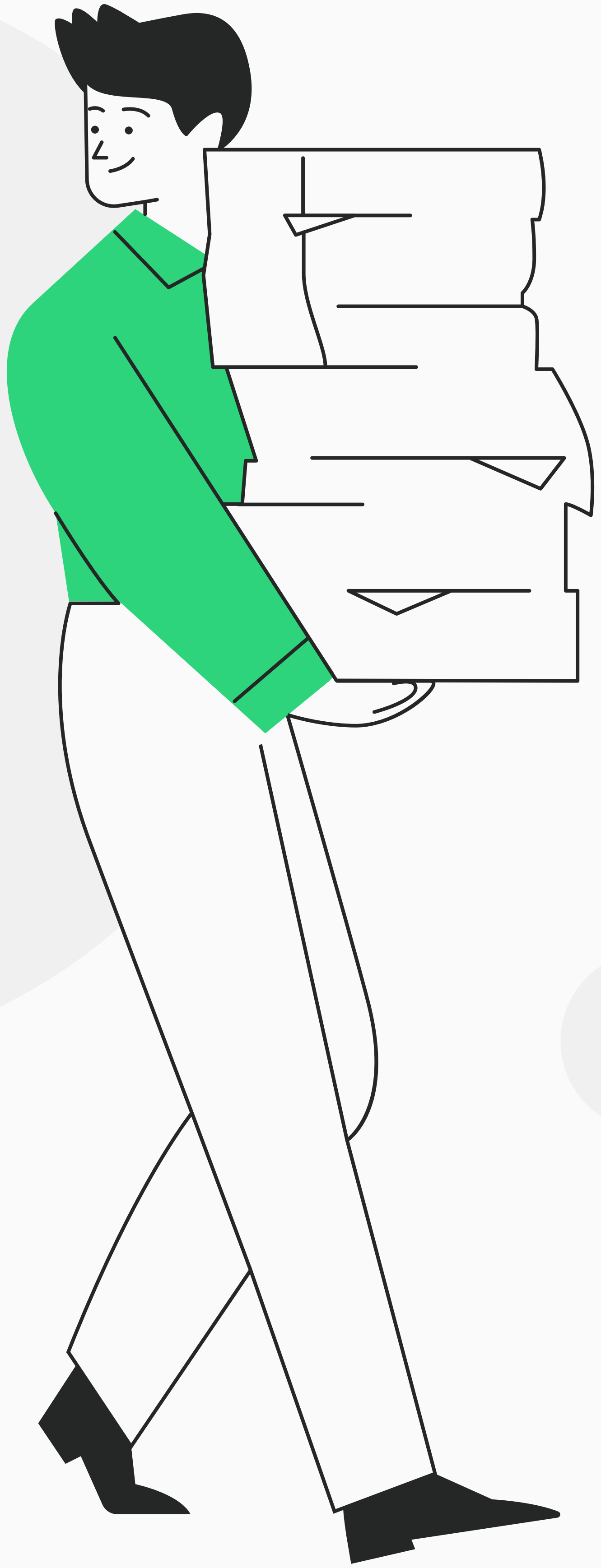
Временная сложность

ДОКАЗАТЕЛЬСТВО

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)

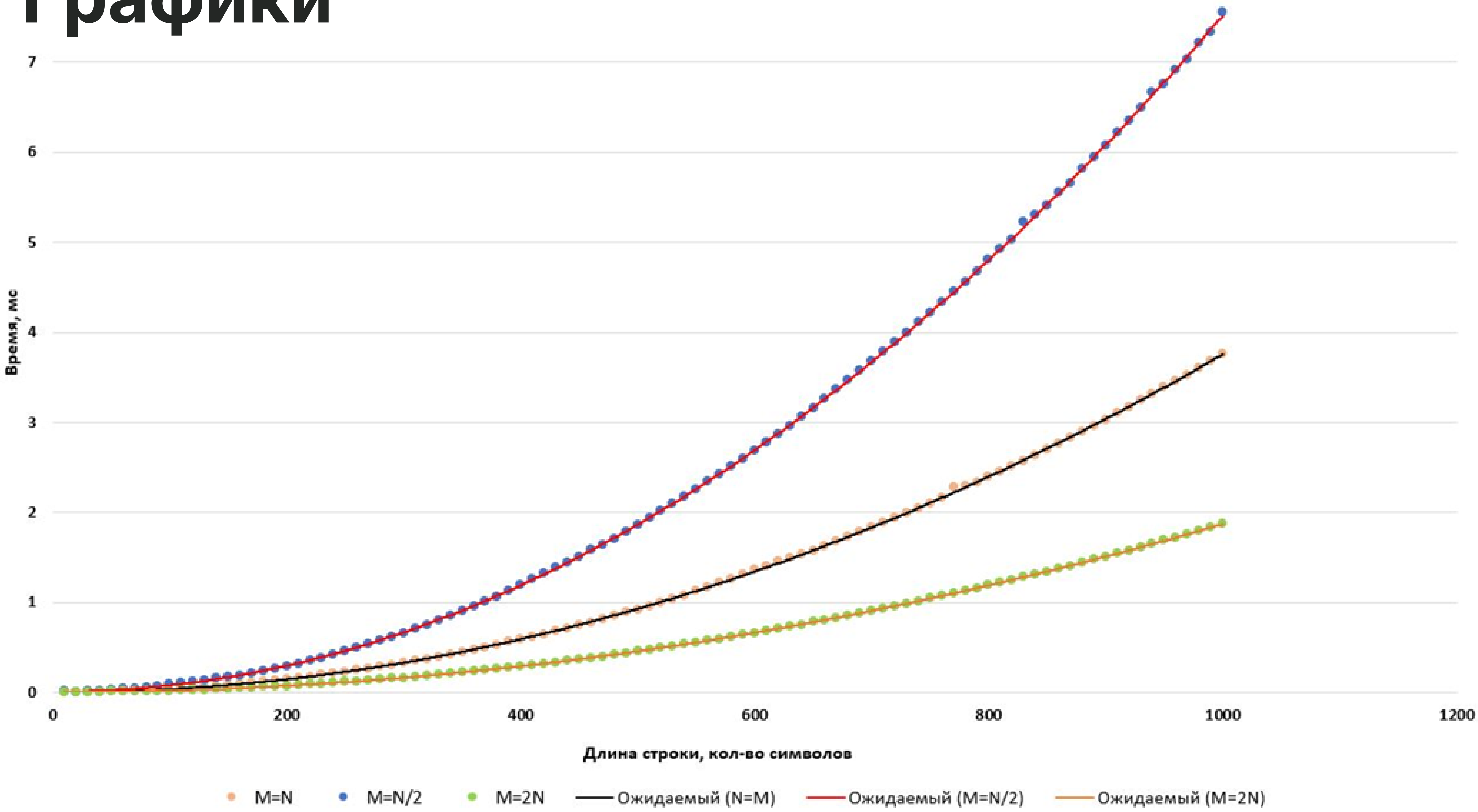
Временная сложность

ДОКАЗАТЕЛЬСТВО



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

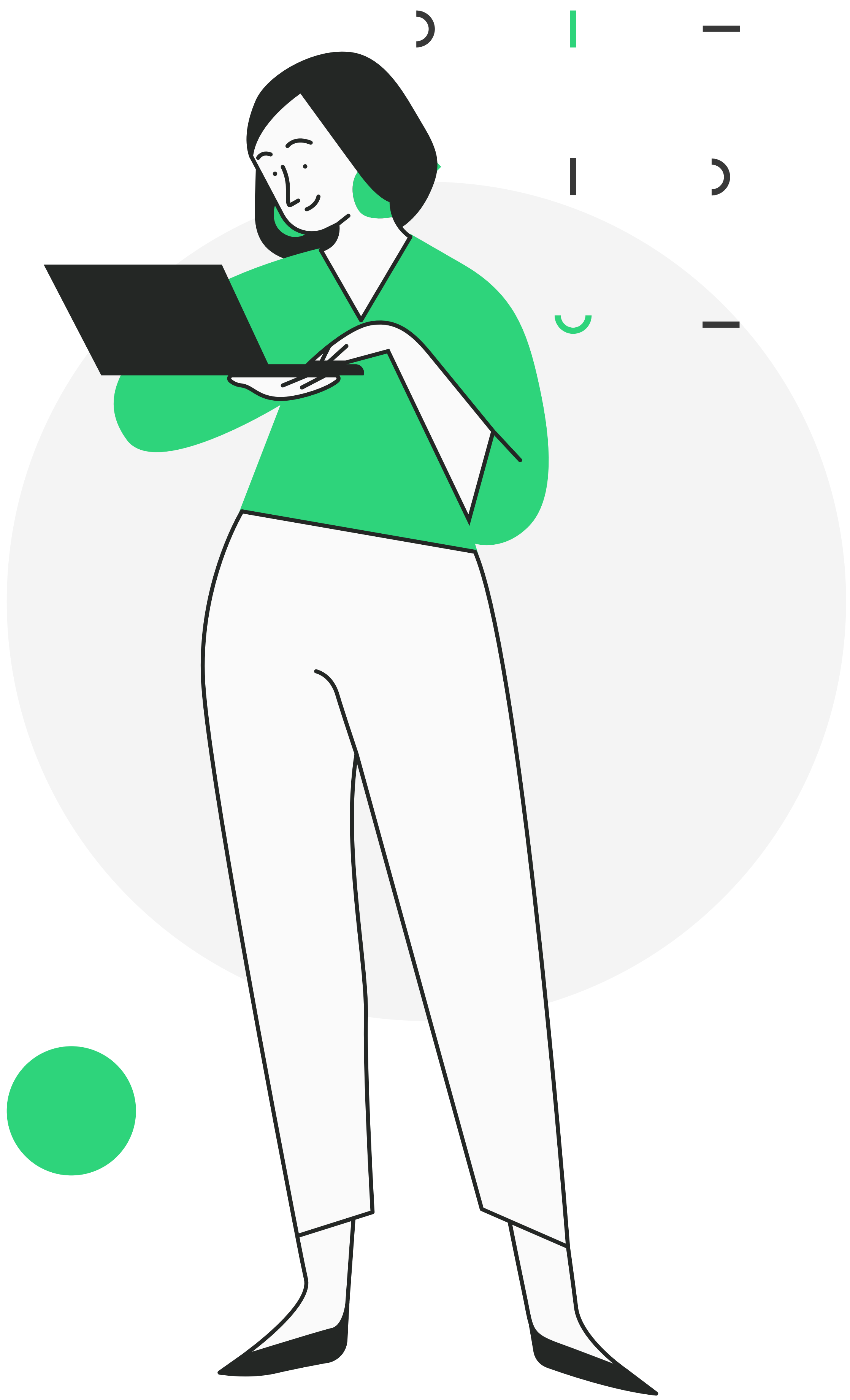
Графики



7
6
5
4
3
2
1
0
Время, мс
0
200
400
600
800
1000
1200
Длина строки, кол-во символов
M=N
M=N/2
M=2N
Ожидаемый (N=M)
Ожидаемый (M=N/2)
Ожидаемый (M=2N)

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))



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

Применимость

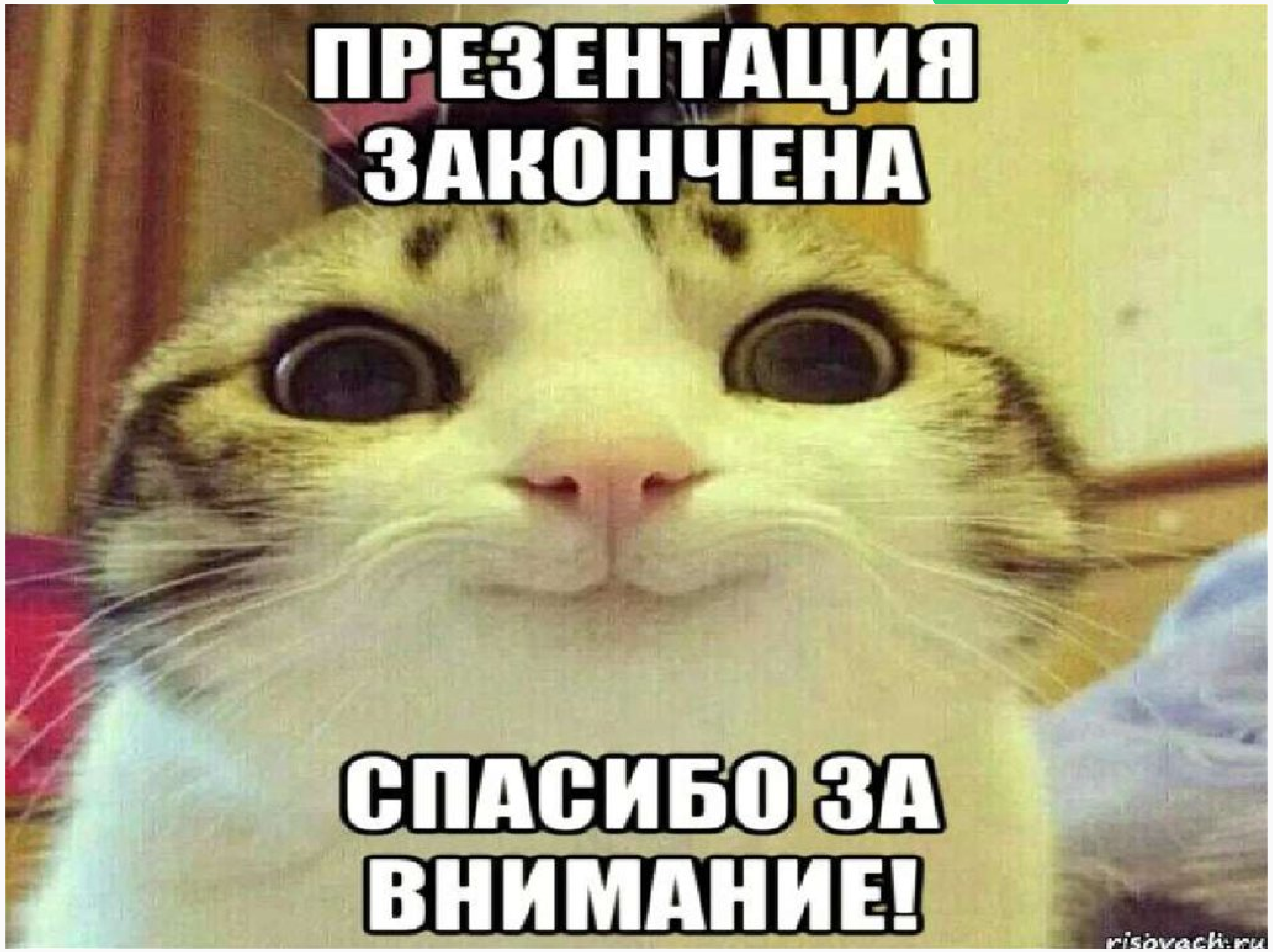
ДЛЯ ИСПРАВЛЕНИЯ ОШИБОК В СЛОВЕ (В ПОИСКОВЫХ СИСТЕМАХ, БД, ПРИ ВВОДЕ ТЕКСТА, ПРИ АВТОМАТИЧЕСКОМ РАСПОЗНАВАНИИ ОТСКАНИРОВАННОГО ТЕКСТА ИЛИ РЕЧИ)

ДЛЯ СРАВНЕНИЯ ТЕКСТОВЫХ ФАЙЛОВ

ДЛЯ ПРОВЕРКИ ТЕКСТОВ НА ПЛАГИАТ

В ЛИНГВИСТИКЕ ДЛЯ ОПРЕДЕЛЕНИЯ НАСКОЛЬКО ЯЗЫКИ ИЛИ ДИАЛЕКТЫ ОТЛИЧАЮТСЯ ДРУГ ОТ ДРУГА

В БИОИНФОРМАТИКЕ ДЛЯ СРАВНЕНИЯ ГЕНОВ, ХРОМОСОМ И БЕЛКОВ


ПРЕЗЕНТАЦИЯ
ЗАКОНЧЕНА
СПАСИБО ЗА
ВНИМАНИЕ!