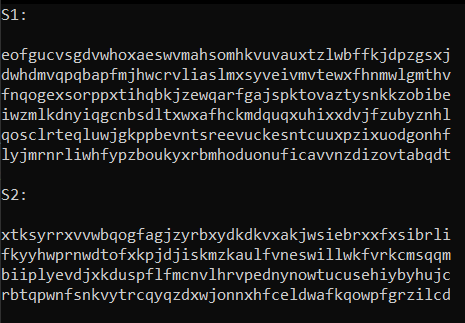
**Лабораторная работа №4**

Динамическое программирование

**Задание 1.**

Для генерации случайных строк используется функция **GenerateRandomString**, которая принимает размер строки и возвращает указатель на массив символов типа **char**. Внутри функции используется функция **rand()** для генерации случайных чисел в диапазоне от 0 до 25 (число букв в латинском алфавите), после чего к результату прибавляется код символа 'a', чтобы получить случайную букву латинского алфавита.

В функции **main()** вызывается функция **GenerateRandomString** для генерации двух строк: **s1** длиной 300 символов и **s2** длиной 200 символов. Затем строки выводятся на экран с помощью цикла **for** и условного оператора **if**, который печатает перенос строки каждые 50 символов.



**Задание 2.**

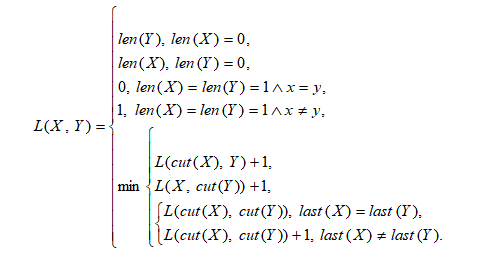
Реализована функция **levenshtein\_r** для вычисления дистанции Левенштейна с помощью рекурсии и функция **levenshtein** для вычисления дистанции Левенштейна с помощью динамического программирования. Для каждого способа вычисления дистанции Левенштейна было измерено время выполнения для различных длин строк.

В основной функции **main** сначала генерируются случайным образом строки **s1** и **s2** заданных длин, а затем измеряется время выполнения для каждого способа вычисления дистанции Левенштейна с помощью функций **levenshtein\_r** и **levenshtein**. Результаты измерений выводятся на экран в виде таблицы, где указывается длина строк и время выполнения для каждого способа.

**Задание 3.**



**Задание 4.**





=5

*=4*



*=4*

*=3*

=3

*=2*

=2

*=1*

=3

*=2*

=2

*=1*

=1

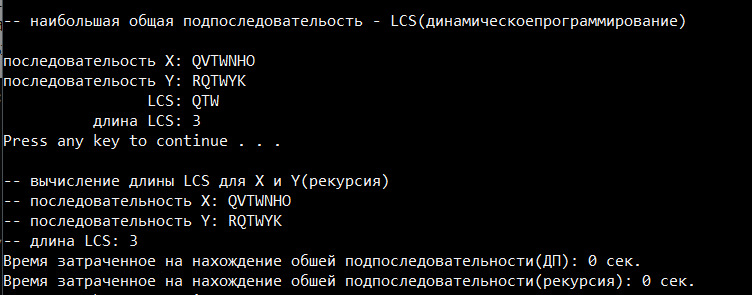
*=1*



**Задание 5.**

**Нечетные варианты**. Выполнить сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения (рекурсивное решение, динамическое программирование). Две последовательности взять в соответствии с вариантом.

Выполнив сравнительный анализ времени затраченного на решение задачи о наибольшей общей подпоследовательности для двух методов решения, можно заметить, что динамический алгоритм работает быстрее, однако, в данном ситуации оба метода затрачивают для выполнения минимально времени. Результаты представлены на рисунке 5:



На рисунке 6 представлен график зависимости от суммарной длинны двух подстрок и необходимом времени выполнения при помощи динамического алгоритма:



**Алгоритм прохождения:**

1. Создаем двумерный массив размером (len(QVTWNHO) + 1) на (len(RQTWYK) + 1), где каждый элемент равен 0.

2. Проходим по каждой строке и столбцу массива, начиная с первого и до последнего.

3. Если значение строки или столбца равно 0, то элемент массива на пересечении этой строки и столбца остается равным 0.

4. Если символ строки и столбца равны, то элемент массива на пересечении этой строки и столбца равен значению элемента на предыдущей диагонали плюс 1, т.е. с\_(𝑛,𝑝) = c\_(𝑛−1,𝑝−1) + 1.

5. Если символ строки и столбца не равны, то элемент массива на пересечении этой строки и столбца равен максимуму между значением элемента на предыдущей строке и значением элемента на предыдущем столбце, т.е. c\_(𝑛,𝑝) = max( с\_(𝑛,𝑝"−1" ), c\_(𝑛−1,𝑝)).

6. После прохода по всем элементам массива, наибольшая общая подпоследовательность будет равна значению элемента на пересечении последней строки и последнего столбца.

**Вывод:** динамический подход к решению задач позволяет выполнять их значительно быстрее, чем рекурсивный, особенно это будет заметно при решении задач с большим объёмом информации, вызвано это тем, что данные кэшируются , а не вычисляются каждый раз заново. Так же, в результате лабораторной работы я сделал следующие выводы:

1. Динамическое программирование - это мощный метод решения задач, позволяющий эффективно решать широкий спектр задач, которые не могут быть решены простыми алгоритмами.

2. Метод динамического программирования заключается в разбиении сложной задачи на более простые подзадачи, решение которых затем комбинируется в общее решение задачи.

3. Решение задач методом динамического программирования отличается высокой скоростью выполнения благодаря использованию кэширования вычислений и быстрой обработке данных.

4. В ходе выполнения лабораторной работы был исследован один из наиболее популярных методов динамического программирования - расстояние Левенштейна. Результаты экспериментов показали, что данная техника может быть очень эффективна для решения задач, связанных с обработкой текстовых данных.