Лабораторная работа 4

ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

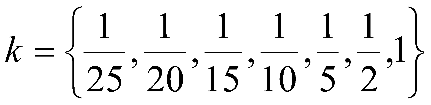
ЦЕЛЬ РАБОТЫ: освоить общие принципы решения задач методом динамического программирования, сравнить полученные решения задач с рекурсивным методом.

ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:

*Задание 1.*

На языке С++ сгенерировать случайным образом строку букв латинского алфавита S1 длиной 300 символов и S2 длиной 200.

*Задание 2.*

Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – дистанцию Левенштейна для , где - длина строки ,  - строка состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

*Задание 3.*

Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).

*Задание 4.*

Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма (в соответствии с вариантом) (каждый шаг алгоритма по примеру из лекции вставить в отчет).

*Задание 5.*

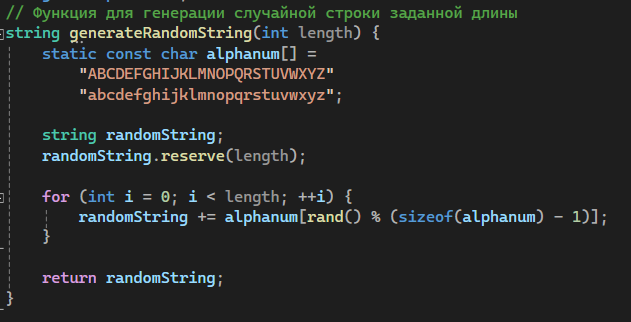
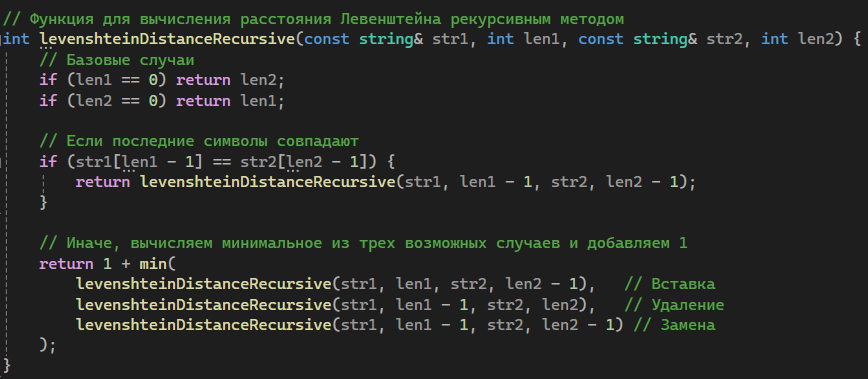
Выполнить сравнительный анализ времени затраченного на решение задачи об оптимальной расстановке скобок при умножении нескольких матриц для двух методов решения (рекурсивное решение, динамическое программирование). Размерность матриц взять в соответствии с вариантом. Объяснить в отчете принцип расставления скобок по итоговой матрице + код + копии экрана.

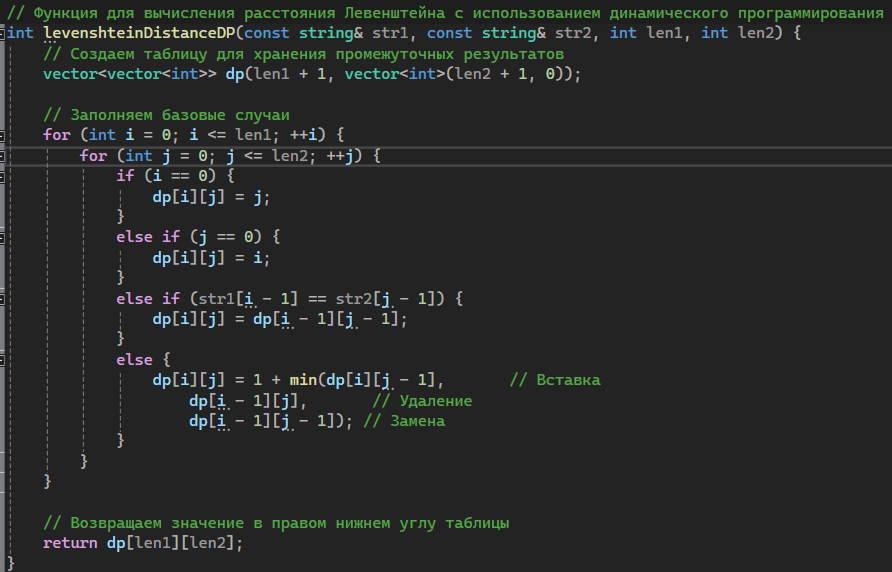
Вариант 2:

|  |  |  |
| --- | --- | --- |
| Том | Исток | 7\*10, 10\*18, 18\*21, 21\*28, 28\*38, 38\*49 |

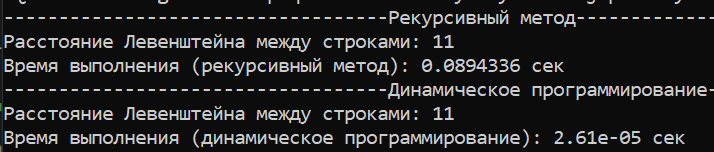
Задание 1:

Функция для генерации случайной строки заданной длины:

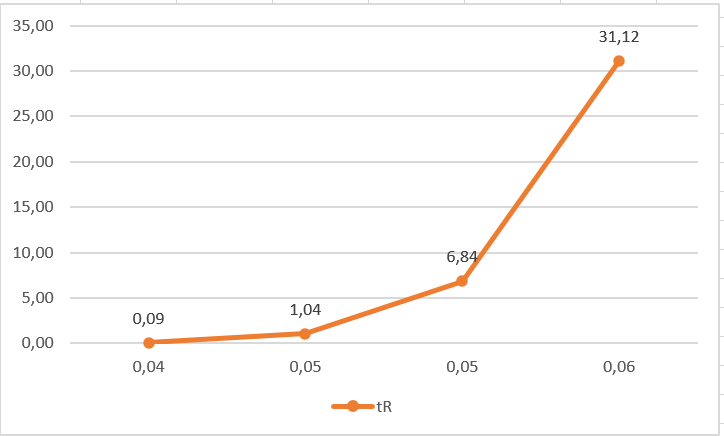
  
Задание 2: 



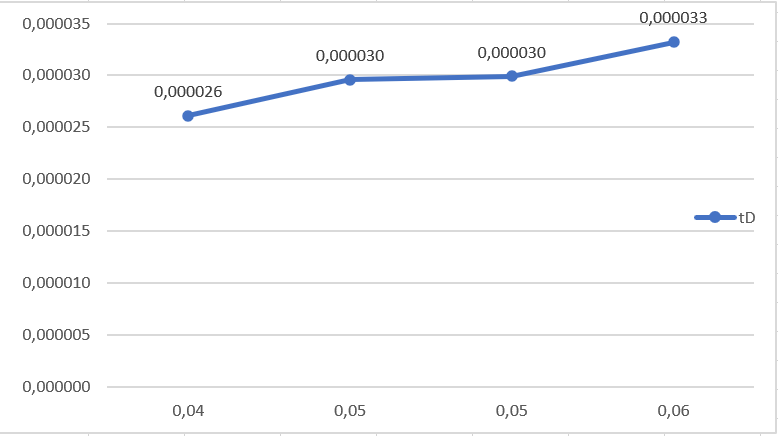
Результат выполнения:



Задание 3:



Экспоненциальная зависимость для рекурсивного алгоритма



Линейная зависимость для динамического программирования

Задание 4:

Для вычисления расстояния Левенштейна между двумя строками методом рекурсии, мы можем использовать следующий подход.

1. Рассмотрим последние символы "м" и "к". Они не совпадают. Мы можем выполнить одно из трех действий:

* Вставить "к" в "Том" и сравнить "Том" с "Исток".
* Удалить "м" из "Том" и сравнить "То" с "Исток".
* Заменить "м" на "к" в "Том" и сравнить "Ток" с "Исток".

1. Если мы вставляем "к" в "Том", то мы сравниваем "Ток" с "Исток". Теперь мы должны снова сравнить две строки "Ток" и "Исток".

* Последние символы "к" и "к" совпадают, так что мы рассматриваем подстроки "То" и "Исто".
* Последние символы "о" и "о" совпадают, так что мы рассматриваем подстроки "Т" и "Ист".

1. Последние символы "Т" и "И" не совпадают. Мы можем выполнить одно из трех действий:

* Удалить "Т" из "Ист" и сравнить пустую строку с "Ист".
* Заменить "Т" на "И" и сравнить пустую строку с "ст".

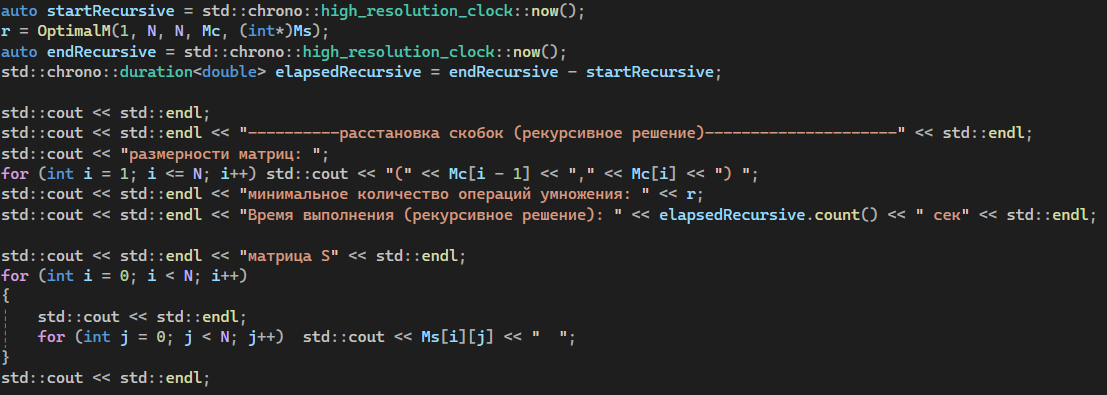
1. Последние символы "" и "И" не совпадают. Мы можем выполнить одно из трех действий:

* Вставить "И" в пустую строку и сравнить пустую строку с "И".
* Удалить "" из "И" и сравнить пустую строку с "И".
* Заменить "" на "И" и сравнить пустую строку с "И".

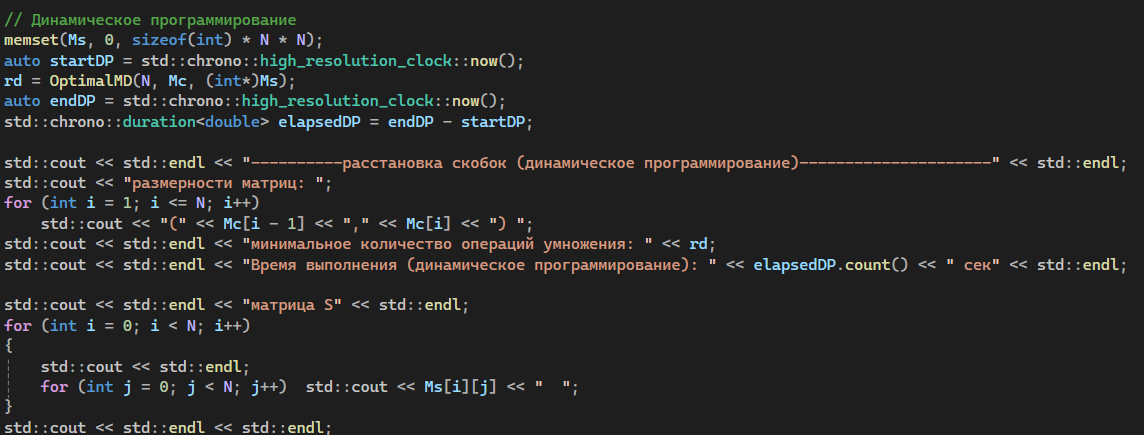
Таким образом, минимальное количество операций, необходимых для превращения строки "Том" в "Исток", равно 4.

Задание 5:

Рекурсивный алгоритм



Динамическое программирование



Задача нахождения расстояния Левенштейна является классической задачей динамического программирования, которая широко применяется в области компьютерной лингвистики, биоинформатики и многих других областях. Основная цель этой задачи - найти минимальное количество операций вставки, удаления и замены, необходимых для преобразования одной строки в другую.

Алгоритм динамического программирования для нахождения расстояния Левенштейна использует матрицу, в которой строки представляют символы первой строки, а столбцы - символы второй строки. Значение в каждой ячейке матрицы соответствует минимальному количеству операций, необходимых для преобразования подстроки первой строки в подстроку второй строки.