**Лабораторная работа 4**

**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

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

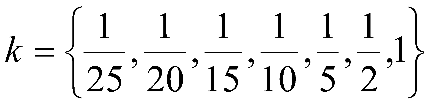
**ЗАДАНИЕ ДЛЯ ВЫПОЛНЕНИЯ:**

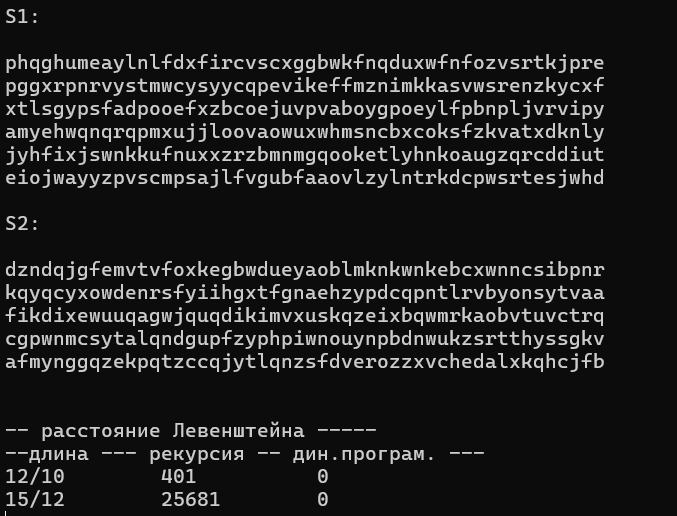
***Задание 1.***

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.***

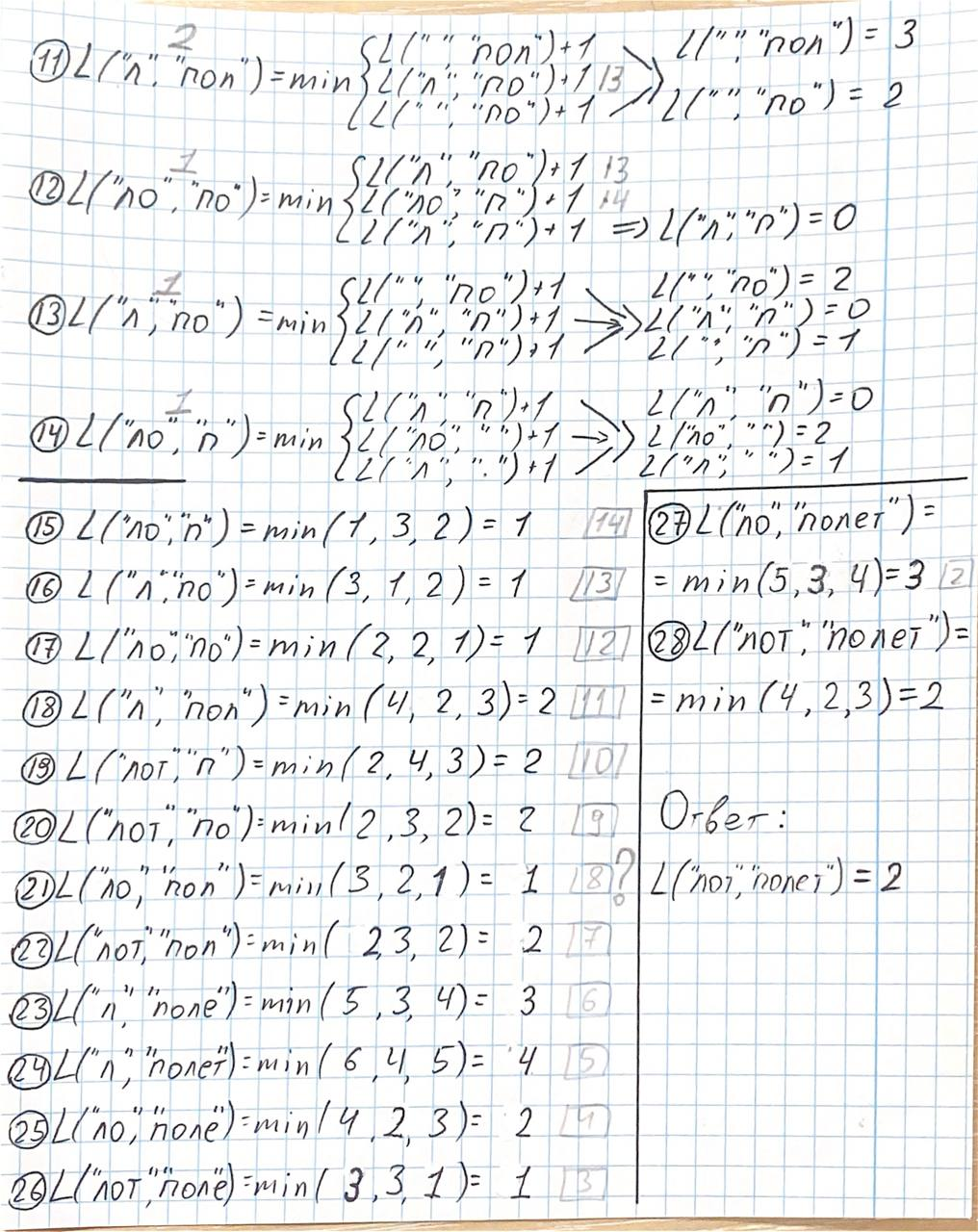
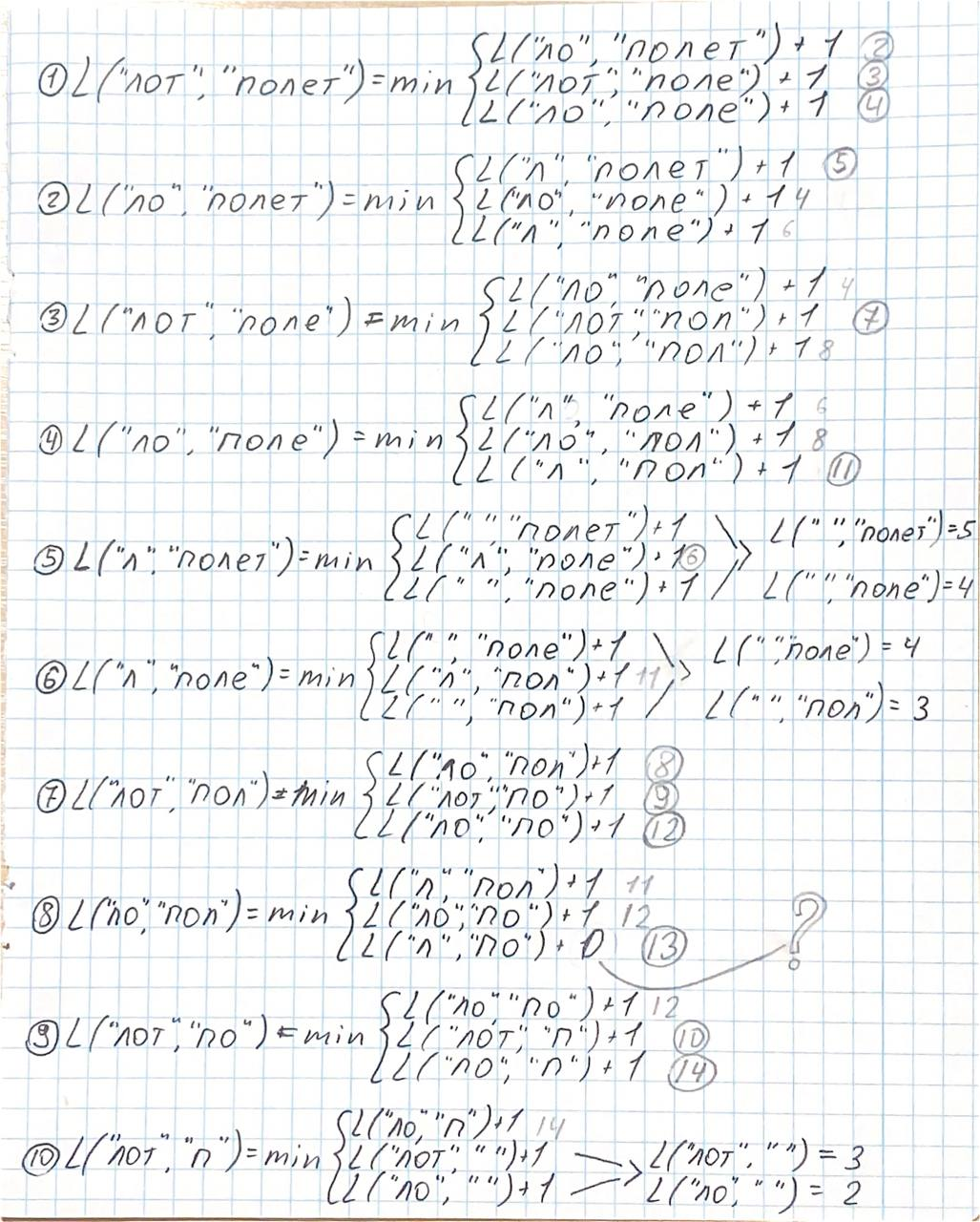
Выполнить сравнительный анализ времени затраченного на вычисление дистанции Левенштейна для двух методов решения. Построить графики зависимости времени вычисления от . (копии экрана и график вставить в отчет).

****

***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).

|  |  |  |
| --- | --- | --- |
| Вариант | Задание 4 | |
| 11 | Лот | Полет |

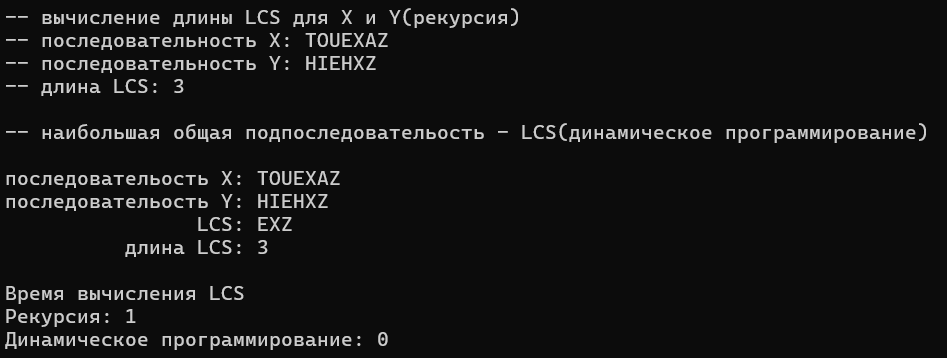


*Результат вычисления дистанции Левенштейна: L(“лот”, “полет”)=2*

***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. Построить графики зависимости времени вычисления от . **Отобразить ход решения в отчете**(по примеру из лекции) + код и копии экрана.

|  |  |  |
| --- | --- | --- |
| Вариант | Задание 5 | |
| 11 | TOUEXAZ | HIEHXZ |



1. **Рекурсивное решение**

Рекурсивный метод уже представлен в коде в lcs. Этот метод использует сравнение последних символов и разветвление на три возможных пути: Если символы совпадают: длина увеличивается на 1, а строки уменьшаются. Если символы разные: выбирается максимальное значение из двух подзадач — либо удаление из первой строки, либо из второй.

1. **Динамическое программирование**

* Метод DP (lcsd) уже описан. Основные шаги:
* Создание двумерного массива для хранения промежуточных результатов.
* Вычисление максимальной длины LCS через итерации.
* Извлечение самой наибольшей подпоследовательности с помощью направления движения по таблице.

В программе для расчёта используется код из методического пособия для вычисления рекурсивного и динамического вычисления дины LCS.

1. **Заполнение и вычисление матриц:**

*1. Левая матрица:* Таблица значений длины LCS.

Эта матрица хранит промежуточные результаты вычисления LCS. Строки и столбцы соответствуют символам двух последовательностей. Значения в ячейках таблицы указывают длину наибольшей общей подпоследовательности между соответствующими подстроками.

*Как заполняется:*

1. Инициализация:

- Первая строка (верхняя) и первый столбец (слева) заполняются нулями, так как, если одна из строк пустая, длина LCS равна 0.

2. Основное правило:

Для каждой ячейки dp[i][j] (где i и j — индексы строки и столбца):

- Если символы строк совпадают (X[i-1] == Y[j-1]):

dp[i][j] = dp[i-1][j-1] + 1;

Добавляем 1 к длине LCS предыдущих подстрок.

- Если символы не совпадают:

dp[i][j] = max(dp[i-1][j], dp[i][j-1]);

Берем максимальное значение между удалением символа из одной строки или другой.

Пример заполнения:

Для строк X = "ALBDACD" и Y = "CDLDCA":

- Столбцы и строки соответствуют символам (с дополнительным "пустым" начальным символом).

- Каждый шаг вычисления учитывает совпадение символов и строит LCS по мере заполнения.

*2. Правая матрица:* Направления движения (Backtracking)

Эта матрица хранит информацию о том, каким образом был вычислен каждый элемент в левой таблице. Она используется для восстановления самой LCS.

Типы стрелок:

- Стрелка вверх (TOP): Значение взято из ячейки сверху (dp[i-1][j]), что означает исключение символа из последовательности X.

- Стрелка влево (LEFT): Значение взято из ячейки слева (dp[i][j-1]), что означает исключение символа из последовательности Y.

- Диагональная стрелка (LEFTTOP): Значение взято из ячейки по диагонали сверху-слева (dp[i-1][j-1]), что указывает на совпадение символов и их включение в LCS.

*Как используется:*

1. Начинаем с последней ячейки (правый нижний угол).

2. Двигаемся по стрелкам:

- Если стрелка диагональная, символ добавляется в LCS.

- Если стрелка вверх или влево, пропускаем соответствующий символ.

Что означают матрицы:

- Левая матрица - показывает числовые значения, отражающие длину LCS для каждой пары подстрок.

- Правая матрица - определяет путь для восстановления самой LCS.

**Вопросы**

* + - 1. В каких областях используется динамическое программирование?

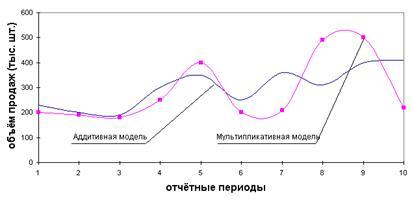
Динамическое программирование используется при оптимальном планировании управляемых процессов и наиболее эффективно в случае многошаговых или многоэтапных процессов принятия решений. Применяется в различных областях, включая компьютерную графику, искусственный интеллект, экономику, теорию управления, биоинформатику и другие.

* + - 1. В чем заключается задача динамического программирования?

Задача динамического программирования заключается в поиске оптимального решения для задачи, которая может быть разбита на более мелкие подзадачи. Решение подзадач сохраняется и используется для поиска решения более крупной задачи.

* + - 1. Чем аддитивная функция отличается от мультипликативной?

Аддитивная функция показывает, какие значения могут быть добавлены друг к другу. Мультипликативная функция показывает, как одно значение может быть умножено на другое, чтобы получить новое значение. На практике отличить аддитивную модель от мультипликативной можно по величине сезонной вариации. Аддитивной модели присуща практически постоянная сезонная вариация, тогда как у мультипликативной она возрастает или убывает, графически это выражается в изменении амплитуды колебания сезонного фактора, как это показано на рисунке



* + - 1. Каков принцип оптимальности Беллмана?

Принцип оптимальности Беллмана заключается в том, что оптимальное решение задачи состоит из оптимальных решений ее подзадач.

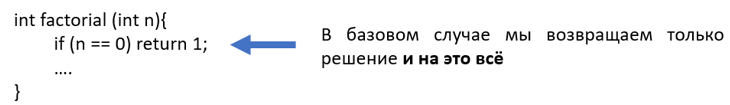
Беллман сказал: “… ***Оптимальное управление должно обладать таким свойством, что каково бы ни было начальное состояние на любом шаге, и управление, выбранное на этом шаге, последующие управления должны выбираться оптимальными относительно состояния, к которому придёт система в конце данного шага”.***

* + - 1. Что такое рекурсивный алгоритм?

Рекурсивный алгоритм - это алгоритм, который использует функцию для решения задачи, вызывая ее саму для решения подзадач.

* + - 1. Что такое рекурсивная функция?

Рекурсивная функция - это функция, которая вызывает саму себя в своем теле. Каждая рекурсивная функция должна иметь базовый случай:



* + - 1. Что такое системный стек?

Системный стек - это структура данных, используемая в компьютерных системах для отслеживания вызовов функций и управления памятью. Кроме того, для хранения контекста (что используется при работе с рекурсивными функциями, т.к. как любой вызов функции связан с сохранением и восстановлением контекста вызывающей функции).

* + - 1. Объясните понятие «глубина рекурсии»

Глубина рекурсии - это количество раз, которое рекурсивная функция вызывает саму себя.

* + - 1. Поясните своими словами схему решения задачи по принципу «разделяй и властвуй»

**Алгоритм** «**разделяй** **и** **властвуй**» рекурсивно разбивает проблему на две или более подзадачи того же или родственного типа, пока они не станут достаточно простыми, чтобы их можно было решить напрямую. Затем решения подзадач объединяются, чтобы дать решение исходной проблемы.

* + - 1. Что такое редакционное расстояние?

Редакционное расстояние - это мера разницы между двумя строками, определяющая минимальное количество операций вставки, удаления и замены символов, необходимых для превращения одной строки в другую.

* + - 1. Что такое подпоследовательность и как её можно получить из последовательности?

Подпоследовательность - это последовательность, полученная путем удаления некоторых элементов из исходной последовательности, сохраняя порядок оставшихся элементов. Это может быть сделано, чтобы найти наибольшую общую подпоследовательность между двумя последовательностями.