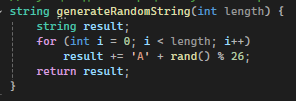
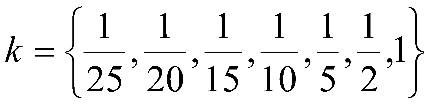
|  |  |  |  |
| --- | --- | --- | --- |
| Вариант | Задание 4 | | Задание 5 |
| 4 | Акр | Якорь | 20\*15, 15\*30, 30\*53, 53\*10, 10\*20, 20\*11 |

***Задание 1.***

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

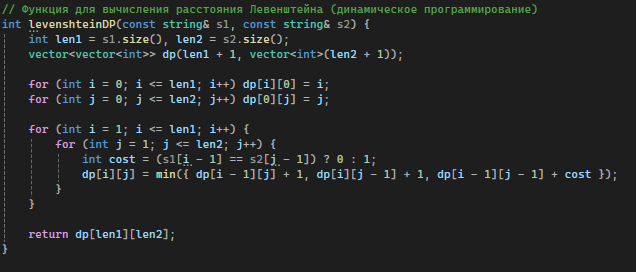
****

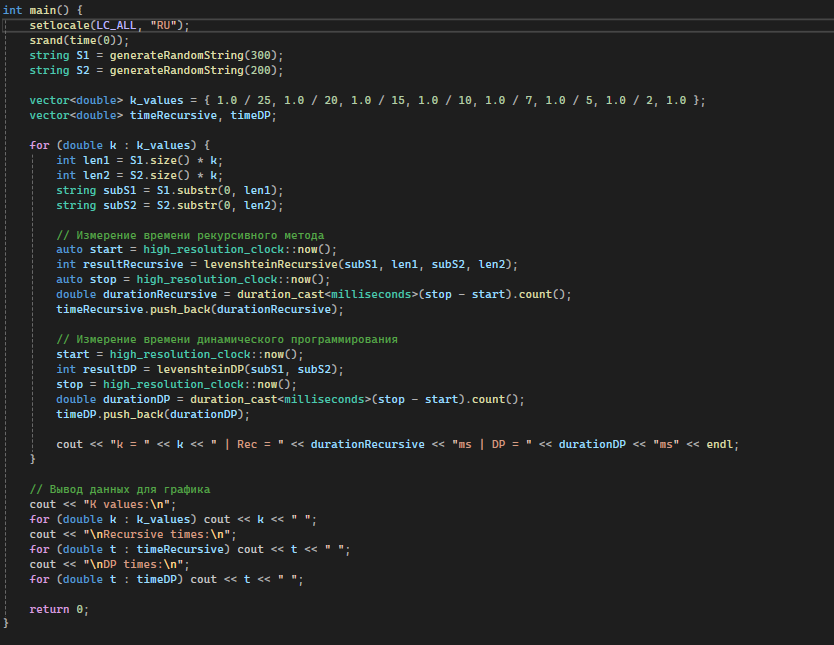
***Задание 2.***

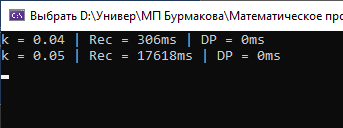
Вычислить двумя способами (рекурсивно и с помощью динамического программирования)  – дистанцию Левенштейна для , где - длина строки ,  - строка состоящая из первых  символов строки . (копии экрана и код вставить в отчет).

**Изображение выглядит как текст, снимок экрана, Шрифт

Контент, сгенерированный ИИ, может содержать ошибки.**

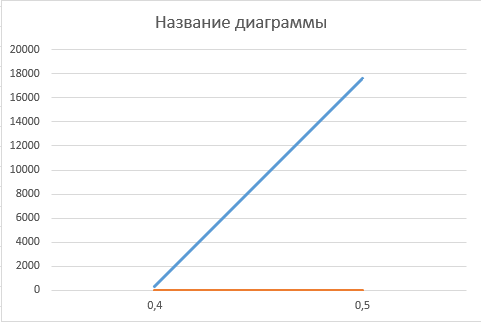
****

****

****

***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).

**Шаг 1: Определение строк**

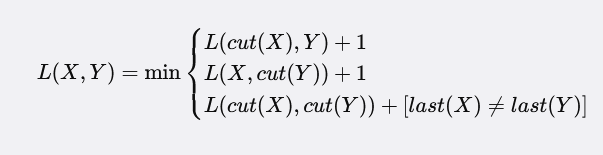
* X = "акр"
* Y = "якорь"

**Шаг 2: Размеры строк**

* len(X) = 3
* len(Y) = 5

**Шаг 3: Рекуррентное соотношение**

Используем следующее рекуррентное соотношение:

* Если len(X) = 0, то L(X, Y) = len(Y)
* Если len(Y) = 0, то L(X, Y) = len(X)
* Если len(X) = len(Y) и X = Y, то L(X, Y) = 0
* Если len(X) = len(Y) и X ≠ Y, то L(X, Y) = 1
* В противном случае: 

**Шаг 4: Применение рекурсии**

**1. Вычисляем L("акр", "якорь")**

* Последние символы: р и ь — разные.
* Вызываем:
  + L("ак","якорь")+1
  + L("акр","якор")+1
  + L("ак","якор")+1

**2. Вычисляем L("ак", "якорь")**

* Последние символы: к и ь — разные.
* Вызываем:
  + L("а","якорь")+1
  + L("ак","якор")+1
  + L("а","якор")+1

**3. Вычисляем L("а", "якорь")**

* Последние символы: а и ь — разные.
* Вызываем:
  + L("","якорь")+1
  + L("а","якор")+1
  + L("","якор")+1

**4. Вычисляем L("", "якорь")**

* L("","якорь")=5 (длина "якорь").

**Возвращаемся к L("а", "якорь")**

* L("а","якорь")=5+1=6

**Возвращаемся к L("ак", "якорь")**

* Теперь вычислим L("ак","якор")

**5. Вычисляем L("ак", "якор")**

* Последние символы: к и р — разные.
* Вызываем:
  + L("а","якор")+1
  + L("ак","яко")+1
  + L("а","яко")+1

**6. Вычисляем L("ак", "яко")**

* Последние символы: к и о — разные.
* Вызываем:
  + L("а","яко")+1
  + L("ак","як")+1
  + L("а","як")+1

**7. Вычисляем L("а", "яко")**

* Последние символы: а и о — разные.
* Вызываем:
  + L("","яко")+1
  + L("а","як")+1
  + L("","як")+1

**8. Вычисляем L("", "яко")**

* L("","яко")=3 (длина "яко").

**Возвращаемся к L("а", "яко")**

* L("а","яко")=3+1=4

**Возвращаемся к L("ак", "яко")**

* Теперь можем вычислить:
  + L("ак","яко")=min(4+1, L("ак","як")+1, L("а","як")+1)

**9. Вычисляем L("ак", "як")**

* Последние символы: к и к — одинаковые.
* Вызываем:
  + L("а","як")

**10. Вычисляем L("а", "як")**

* Последние символы: а и к — разные.
* Вызываем:
  + L("","як")+1
  + L("а","я")+1
  + L("","я")+1

**11. Вычисляем L("", "як")**

* L("","як")=2

**Возвращаемся к L("а", "як")**

* L("а","як")=2+1=3

**Заключение**

Теперь возвращаемся к L("ак", "якорь"):

* Итоговая дистанция Левенштейна:
  + L("акр","якорь")=min(6,4,3)+1=3

***Задание 5.***

**Четные варианты**. 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.

Задача о расстановке скобок при умножении матриц заключается в том, чтобы минимизировать количество операций умножения матриц. Рассмотрим два метода решения: рекурсивный подход и динамическое программирование. В ходе анализа мы будем использовать следующие размерности матриц:

* A1: 20 x 15
* A2: 15 x 30
* A3: 30 x 53
* A4: 53 x 10
* A5: 10 x 20
* A6: 20 x 11

**Принцип расстановки скобок**

Оптимальная расстановка скобок определяет, как сгруппировать матрицы для минимизации количества операций. Основная идея заключается в том, чтобы разбить последовательность матриц на подзадачи, вычисляя минимальные стоимости для каждой подзадачи. В конечном итоге, мы получаем матрицу, которая показывает, где ставить скобки.

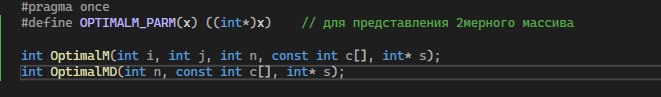
**Методы решения**

**1. Рекурсивный метод**

Этот метод использует рекурсию для вычисления минимального количества операций. Он имеет экспоненциальную сложность O(2n)O(2n), что делает его неэффективным для больших n.

**2. Динамическое программирование**

Этот метод использует таблицу для хранения результатов подзадач, что позволяет избежать повторных вычислений. Сложность этого метода составляет O(n3)O(n3), что значительно быстрее для больших n.

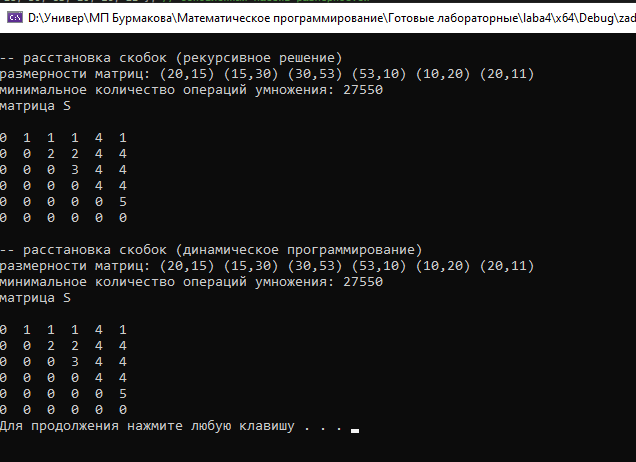


Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, программное обеспечение

Контент, сгенерированный ИИ, может содержать ошибки.

Вот результат выполнения программы:  


Время (рекурсивное решение): 2.2e-06 секунд (2.2 мкс)

Время (динамическое программирование): 2.8e-06 секунд (2.8 мкс)

А1 \* А2 \* А3 \* А4 \* А5 \* А6

По матрице сделаем вывод, как будут расставлены скобки:

s[1][6] = 1  
(А1 \* (А2 \* А3 \* А4 \* А5 \* А6))

s[2][6] = 4

(А1 \* ((А2 \* А3 \* А4) \* (А5 \* А6)))

s[2][4] = 2

(А1 \* ((А2 \* (А3 \* А4)) \* (А5 \* А6)))

ИТОГ:

**(А1 \* ((А2 \* (А3 \* А4)) \* (А5 \* А6)))**