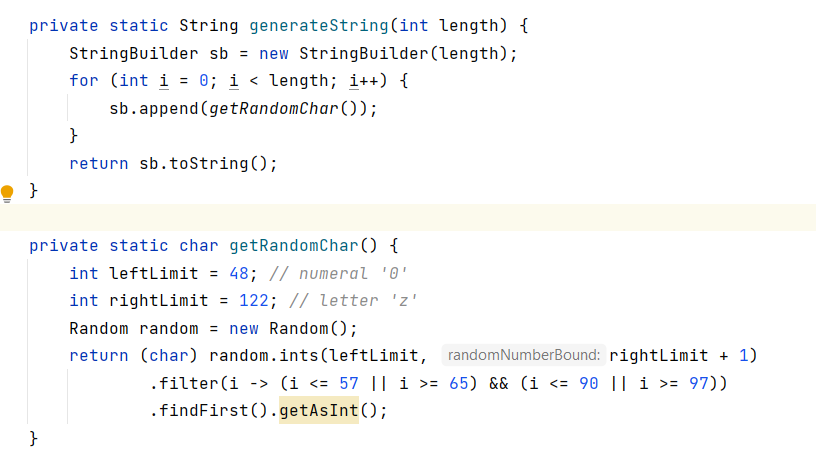
**Лабораторная работа №4. Динамическое программирование**

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

**Ход работы**

## **1. Реализация генератора случайных строк.**

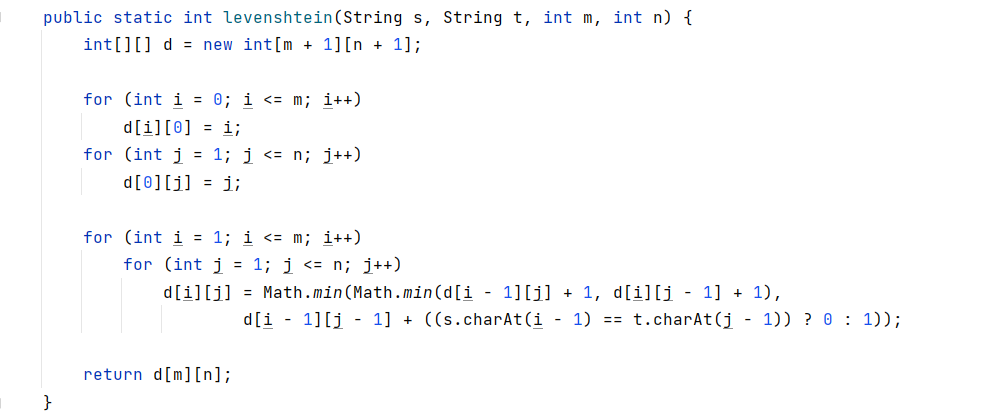
Генератор случайных строк был реализован на языке программирования Java, код программы приведен в листинге 1.1.



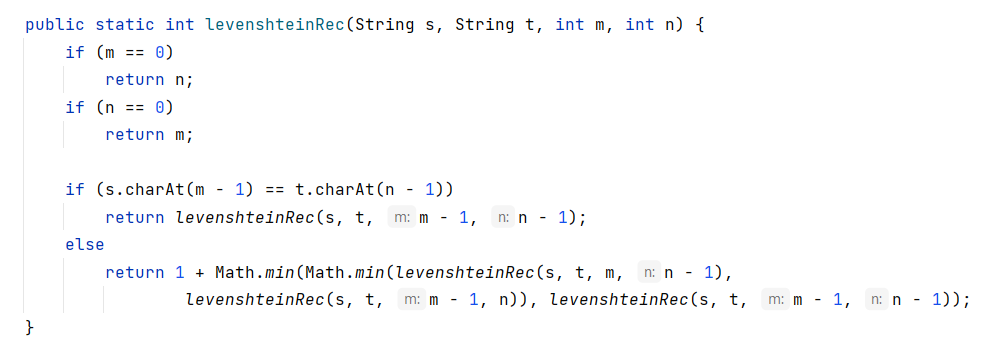
Листинг 1.1. Функция генерации случайной строки

## **2. Вычисление дистанции Левенштейна.**

Функции вычисления дистанции Левенштейна, используя рекурсию и динамическое программирование, были написаны на языке программирования Java. Код функций приведен в листингах 2.1 и 2.2.



Листинг 2.1. Функция Левенштейна, используя динамическое программирование



Листинг 2.2. Функция Левенштейна, используя рекурсия

## **3. Сравнительный анализ двух методов вычисления.**

В тестах были использованы две строки длиной 300 и 250 символов. В функции передавались укороченные подстроки, которые увеличивались на каждой итерации цикла. Для рекурсивного алгоритма использовались только первые две итерации, для динамического – все. Результаты тестов приведены на рисунке 3.1.

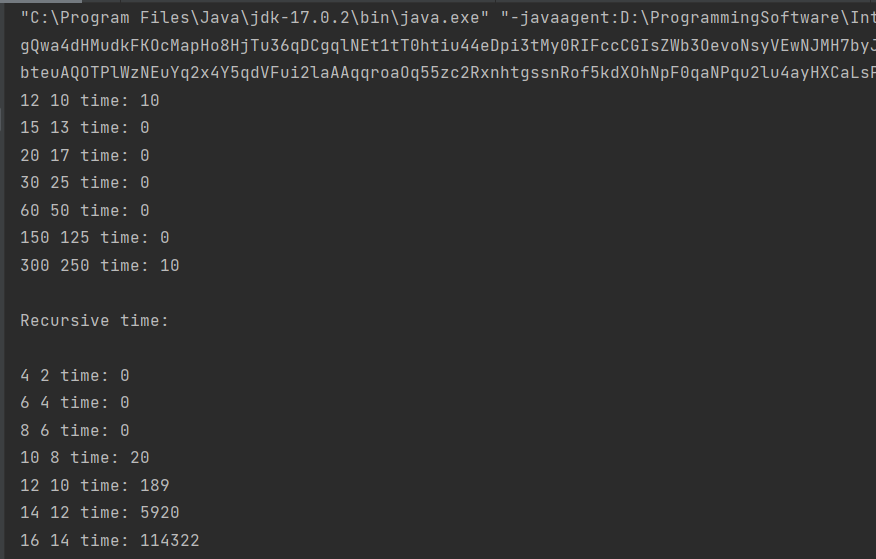


Рисунок 3.1. Результаты тестов двух функций

График, построенный на результате тестов приведен на рисунке 3.2.

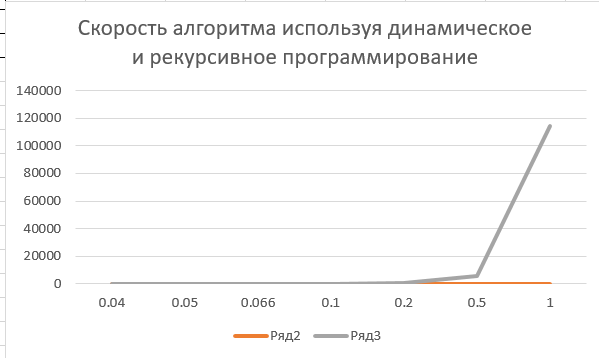
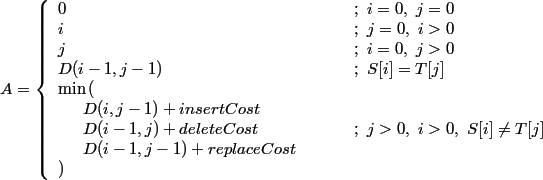


Рисунок 3.2. График времени выполнения алгоритмов

## **4. Пример вычисления дистанции Левенштейна.**

1 вариант: «Бес», «Блеск».



**Решение:**

1. L(«Бес», «Блеск») = min

2. L(«Бе», «Блеск») = min

3. L(«Бес», «Блес») = min

4. L(«Бе», «Блес») = min

5. L(«Б», «Блеск») = min

L(«», «Блеск») = 5,

L(«», «Блес») = 4

7. L(«Б», «Блес») = min

L(«», «Блес») = 4,

L(«», «Бле») = 3

8. L(«Бес», «Бле») = min

9. L(«Бе», «Бле») = min

10. L(«Б», «Бле») = min

L(«», «Бле») = 3,

L(«», «Бл») = 2

11. L(«Бес», «Бл») = min

12. L(«Бе», «Бл») = min

L(«Б», «Б») =0

13. L(«Б», «Бл») = min

L(«», «Бл») =2

L(«», «Б») = 1

L(«Б», «Б») =0

14. L(«Бес», «Б») = min

L(«Бес», «») =3,

L(«Бе», «») = 2

15. L(«Бе», «Б») = min

L(«Бе», «») =2,

L(«Б», «») = 1

L(«Б», «Б») =0

16. L(«Бе», «Б») = min (1, 3, 2) = 1

17. L(«Бес», «Б») = min (2, 4, 3) = 2

18. L(«Б», «Бл») = min (3, 1, 2) = 1

19. L(«Бе», «Бл») = min (2, 2, 1) = 1

20. L(«Бес», «Бл») = min (2, 3, 2) = 2

21. L(«Б», «Бле») = min (4, 2, 3) = 2

22. L(«Бе», «Бле») = min (3, 2, 2) =2

23. L(«Бес», «Бле») = min (2, 3, 2) = 2

24. L(«Б», «Блес») = min (5, 3, 4) = 3

25. L(«Б», «Блеск») = min (6, 4, 5) = 4

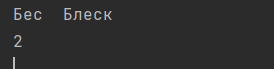
26. L(«Бе», «Блес») = min (4, 2, 3) = 2

27. L(«Бес», «Блес») = min (3, 3, 2) = 2

28. L(«Бе», «Блеск») = min (5, 4, 4) = 4

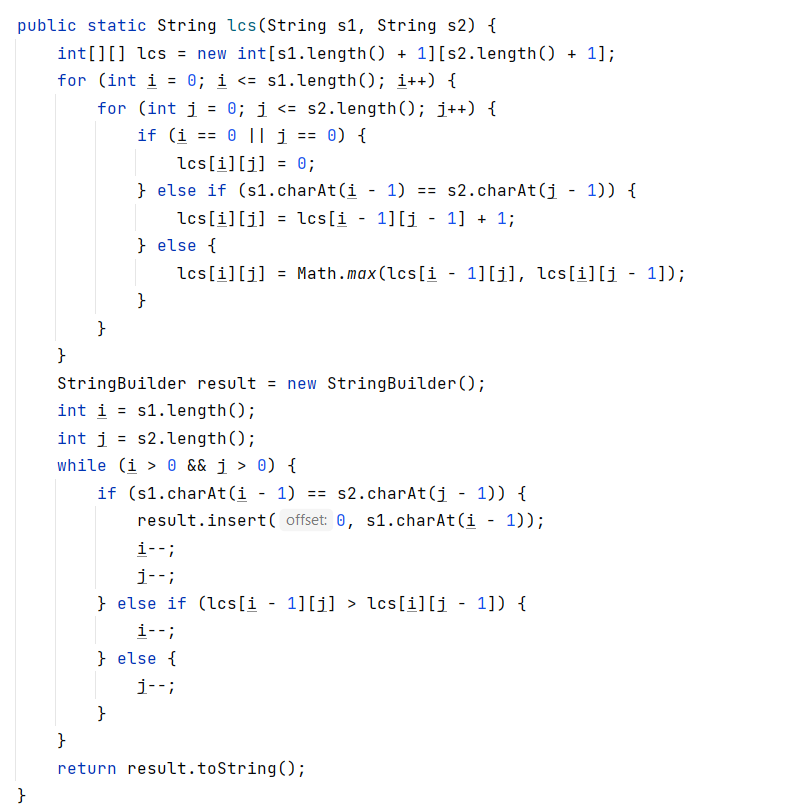
29. L(«Бес», «Блеск») = min (4, 2, 3) = 2

Дистанция Левенштейна равна 2.

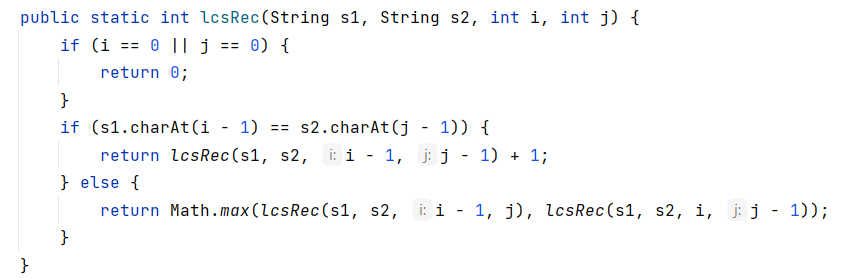


## **5. Решение и анализ задачи о наибольшей общей последовательности.**

Для решения задачи были реализованы функции на языке Java, которые вычисляют наибольшую общую последовательность двумя способами, которые приведены в листингах 5.1 и 5.2.



Листинг 5.1. Реализация алгоритма с использованием динамического программирования



Листинг 5.2. Реализация алгоритма с использованием рекурсии

Результат выполнения программы для нахождения наибольшей общей последовательности двух строк обоими алгоритмами приведён на рисунке 5.1.

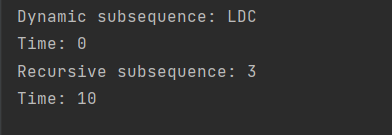


Рисунок 5.1. Результат выполнения задачи о поиске наибольшей общей последовательности

Как можно заметить, время выполнения при малом размере строк отличается незначительно, на рисунке 5.2. приведен график зависимости скоростей.

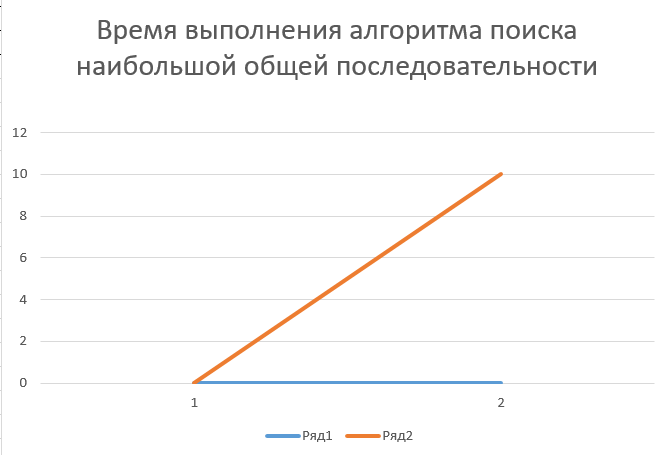


Рисунок 5.2. Время выполнения алгоритмов

|  |  |
| --- | --- |
| ALBDACD | CDLDCA |

Рисунок 5.3. Матрица решения задачи