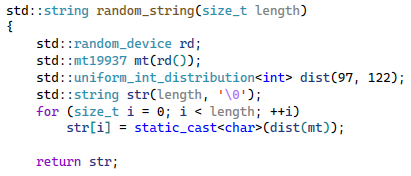
**Лабораторная работа №4. Динамическое программирование**

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

**Ход работы**

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

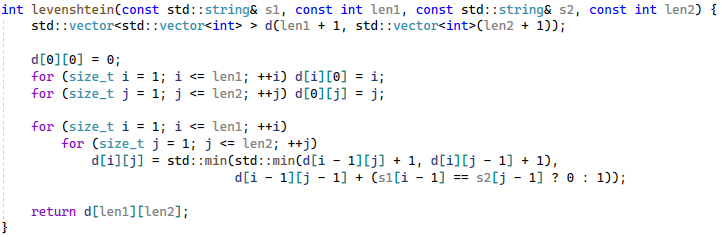
Генератор случайных строк был реализован на языке программирования C++, код программы приведен в листинге 1.1.



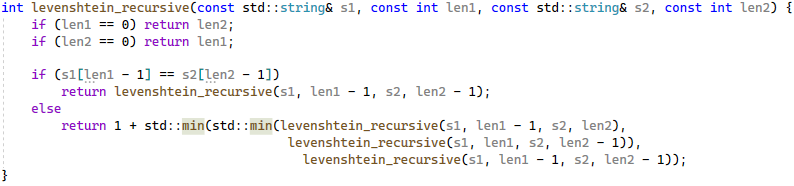
Листинг 1.1. Функция генерации случайной строки

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

Функции вычисления дистанции Левенштейна, используя рекурсию и динамическое программирование, были написаны на языке программирования С++. 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.

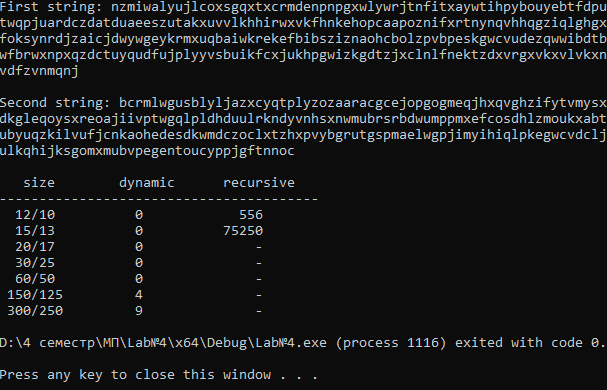


Рисунок 3.1. Результаты тестов двух функций

График, построенный на результате тестов приведен на рисунке 3.2.

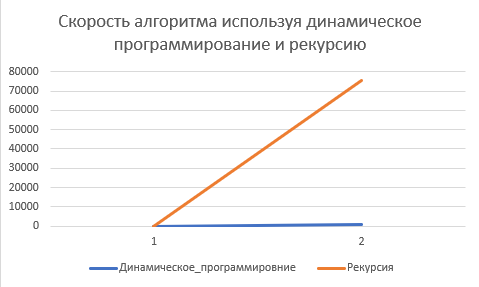
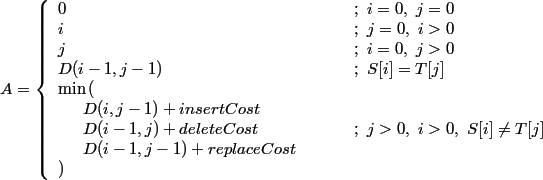


Рисунок 3.2. График времени выполнения алгоритмов

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

9 вариант: «Вол», «Колун».



**Решение:**

1. L(«Вол», «Колун») = min

2. L(«Во», «Колун») = min

3. L(«Вол», «Колу») = min

4. L(«Во», «Колу») = min

5. L(«В», «Колун») = min

L(«», «Колун») = 5,

L(«», «Колу») = 4

6. L(«В», «Колу») = min

L(«», «Кол») = 3

7. L(«Вол», «Кол») = min

8. L(«Во», «Кол») = min

9. L(«В», «Кол») = min

L(«», «Ко») = 2

10. L(«Вол», «Ко») = min

11. L(«Во», «Ко») = min

12. L(«В», «Ко») = min

L(«», «К») = 1

13. L(«Вол», «К») = min

L(«Вол», «») = 3

14. L(«Во», «К») = min

L(«Во», «») = 2,

L(«В», «») = 1

15. L(«В», «К») = min

16. L(«Во», «К») = min (2, 3, 2) = 2

17. L(«Вол», «К») = min (4, 4, 2) = 2

18. L(«В», «Ко») = min (3, 2, 2) = 2

19. L(«Во», «Ко») = min (4, 4, 3) = 3

20. L(«Вол», «Ко») = min (4, 3, 3) = 3

21. L(«В», «Кол») = min (4, 3, 3) = 3

22. L(«Во», «Кол») = min (4, 4, 3) = 3

23. L(«Вол», «Кол») = min (3, 3, 2) = 2

24. L(«В», «Колу») = min (5, 4, 4) = 4

25. L(«В», «Колун») = min (6, 5, 5) = 5

26. L(«Во», «Колу») = min (5, 3, 4) = 3

27. L(«Вол», «Колу») = min (4, 2, 3) = 2

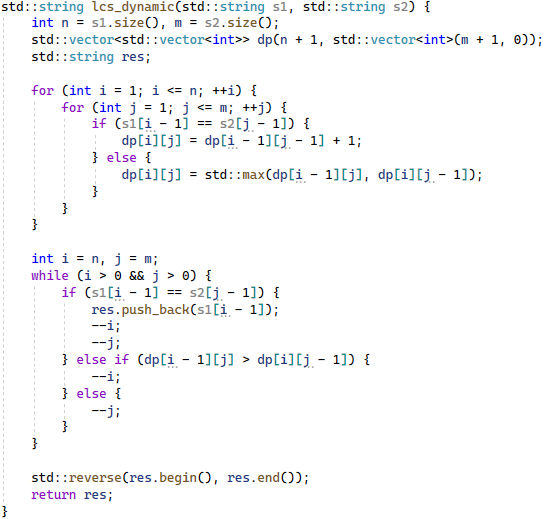
28. L(«Во», «Колун») = min (6, 4, 4) = 4

29. L(«Вол», «Колун») = min (4, 3, 4) = 3

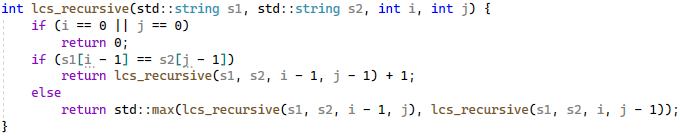
Дистанция Левенштейна равна 3.

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

Для решения задачи были реализованы функции на языке C++, 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.

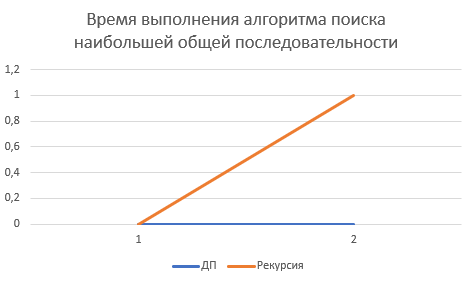


Рисунок 5.2. Время выполнения алгоритмов

Матрица решения задачи приведена на рисунке 5.3.

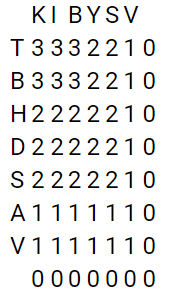


Рисунок 5.3. Матрица решения задачи