**Тараканов Никита Сергеевич**

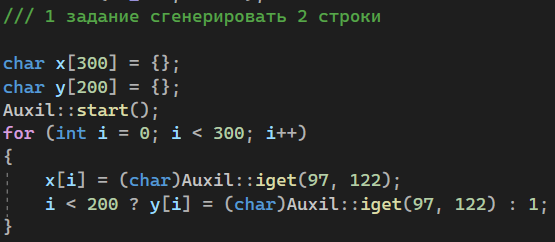
**2 курс 4 группа ПОИТ**

**Отчет**

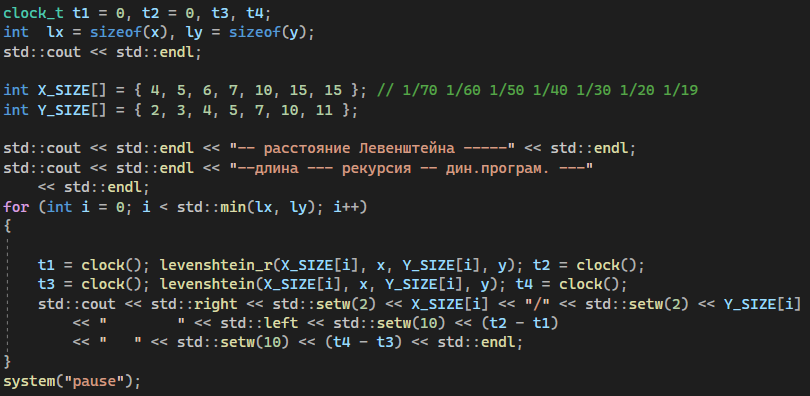
**ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ**

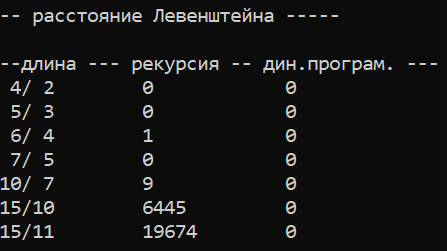
**1 задание**

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



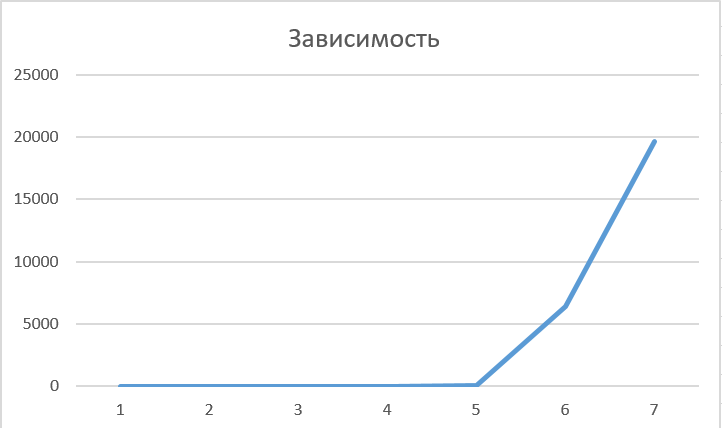
**2 задание**



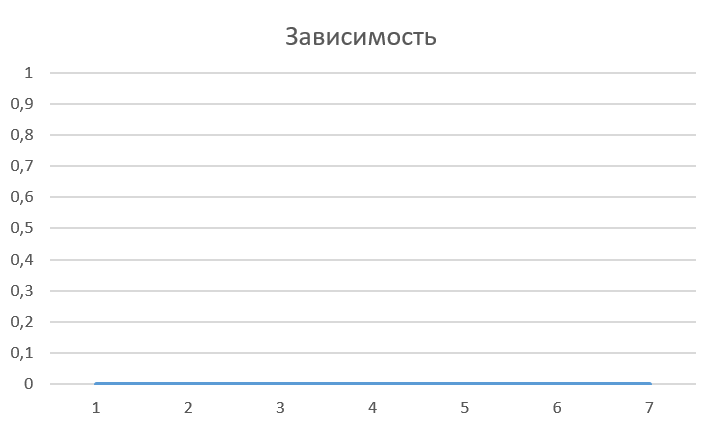


**3 задание**

Для рекурсии:



Для ДП:



**4 задание**

Реализовать вручную пример вычисления дистанции Левенштейна при помощи рекурсивного алгоритма:

S1 = Раб, S2 = Барка



В предыдущем выражении используются символы   и  Разъясним их смысл:

 – количество символов в заданной строке. Например, 

 – заданная строка без последнего символа. Например, 

 – последний символ заданной строки. Например, 

Выполнение задания:

1. L(“Раб”, “Барка”) =

2. L(“Ра”, “Барка”) =

3. L(“Раб”, “Барк”) =

4. 1. L(“Ра”, “Барк”) =

5. L(“Р”, “Барка”) =

6. L(“Р”, “Бар”) =

7. L(“Раб”, “Бар”) =

8. L(“Ра”, “Бар”) =

9. L(“Р”, “Бар”) =

10. L(“Р”, “Барк”) =

,

11. L(“Р”, “Ба”) =

12. L(“Раб”, “Ба”) =

13. L(“Ра”, “Ба”) =

L(“Р”, “Б”) = 1,

14. L(“Раб”, “Б”) =

15. L(“Ра”, “Б”) =

, ,

16. L(“Ра”, “Б”) = min(2, 3, 2) = 2.

17. L(“Раб”, “Б”) = min(3, 4, 3) = 3.

18. L(“Ра”, “Ба”) = min(3, 3, 1) = 1.

19. L(“Р”, “Ба”) = min(3, 2, 2) = 2.

20. L(“Раб”, “Ба”) = min(2, 4, 3) = 2.

21. L(“Р”, “Барк”) = min(5, 4, 4) = 4.

22. L(“Р”, “Бар”) = min(4, 3, 3 ) = 3.

23. L(“Ра”, “Бар”) = min(3, 2 , 3) = 2.

24. L(“Раб”, “Бар”) = min(3, 3,2 ) = 2 .

25. L(“Р”, “Барка”) = min(6, 5, 5) = 5.

26. L(“Ра”, “Барк”) = min(4, 3 , 3) = 3 .

27. L(“Раб”, “Барк”) = min(4, 3, 3) = 3.

28. L(“Ра”, “Барка”) = min(6, 4 , 4) = 4.

29. L(“Раб”, “Барка”) = min(5, 4, 4) = 4.

Редакционное расстояние равно 4.

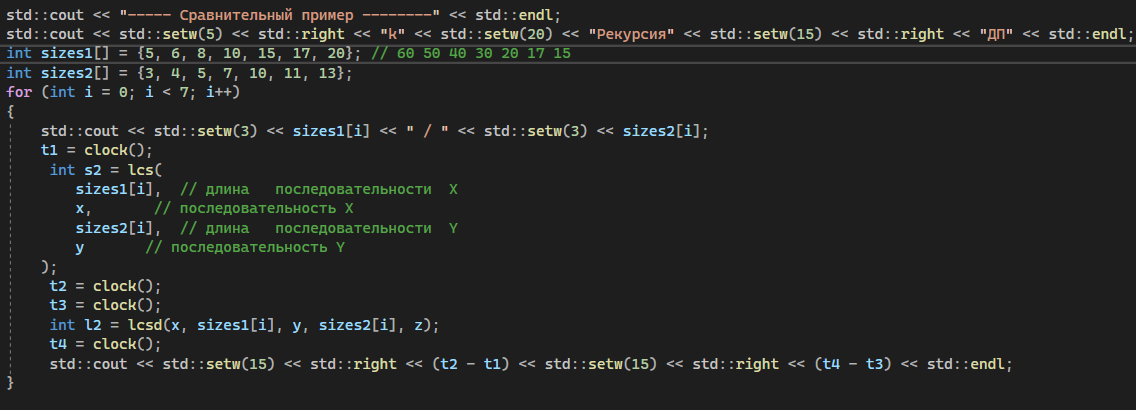


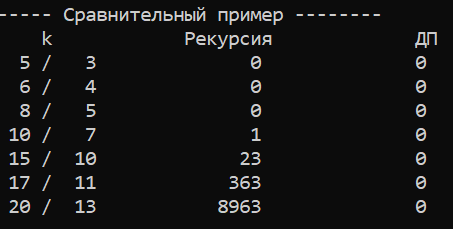
**Задание 5**

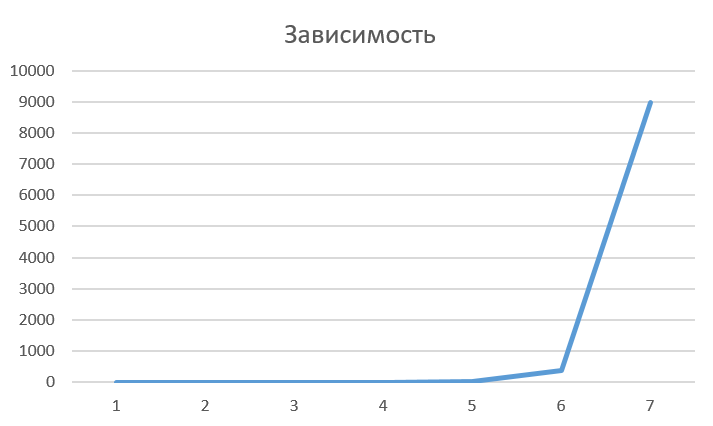
**13 вариант – нечетное задание**

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 .

S1 = BXWAFRE, S2 = XCDUFR







**Вопросы для защиты:**

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

Динамическое программирование - это метод решения задач оптимизации,

который основывается на принципе разбиения их на более мелкие подзадачи и нахождения оптимального решения путем комбинации оптимальных решений подзадач. Он находит широкое применение в различных областях, включая:

- Алгоритмы оптимизации маршрутов и планирования, такие как задача коммивояжера, оптимизация расписания, нахождение кратчайшего пути в графах.

- Обработка сигналов и обработка изображений, например, для поиска оптимального маршрута для передачи данных в сети связи.

- Машинное обучение и искусственный интеллект, например, для поиска оптимальных параметров в алгоритмах машинного обучения.

- Финансы и экономика, например, для оптимизации портфелей инвестиций и управления рисками.

- Биология и генетика, например, для поиска оптимальных последовательностей генов и молекул ДНК.

- Кибербезопасность, например, для анализа и защиты от кибератак.

- Инженерия, например, для оптимизации конструкций и управления производственными процессами.

- Комбинаторика, например, для нахождения оптимальных решений задач

сочетания и перестановки.

В целом, динамическое программирование может применяться в любой области,

где необходимо решить задачу оптимизации, которая может быть разбита на более мелкие подзадачи.

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

Задача динамического программирования заключается в нахождении оптимального решения задачи оптимизации путем разбиения ее на более мелкие подзадачи и нахождения оптимальных решений для каждой из них.

Основной идеей динамического программирования является сохранение результатов вычислений для использования в последующих вычислениях. Это позволяет избежать повторных вычислений и ускорить работу алгоритма.

Для решения задачи динамического программирования необходимо выполнить следующие шаги:

- Определить структуру оптимального решения задачи, которая может быть разбита на более мелкие подзадачи.

- Определить функцию оптимальности, которая определяет оптимальное решение для каждой подзадачи.

- Разработать алгоритм для вычисления оптимальных решений для каждой подзадачи. Это может быть выполнено с помощью рекурсивной или итеративной процедуры.

- Определить начальные условия для решения задачи и вычислить оптимальное решение.

- При необходимости можно использовать полученное оптимальное решение для решения более крупных задач.

В общем случае, задача динамического программирования заключается в нахождении оптимального решения для заданной задачи оптимизации с помощью разбиения ее на более мелкие подзадачи и нахождения оптимальных решений для каждой из них.

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

В динамическом программировании, аддитивная функция используется для решения задач на кратчайший путь, когда затраты на переход между узлами зависят только от самого перехода и не зависят от других переходов. Например, если мы ищем кратчайший путь между двумя узлами в графе, то затраты на переход от одного узла к другому будут равны вне зависимости от того, какой путь мы выбираем. В этом случае, мы можем использовать аддитивную функцию для расчета минимальной стоимости перехода между узлами.

С другой стороны, мультипликативная функция используется в задачах на максимальный поток и минимальный разрез, когда затраты на переход зависят от других переходов. Например, если мы ищем максимальный поток между двумя узлами в графе, то затраты на переход от одного узла к другому будут зависеть от того, сколько потока уже прошло через другие узлы. В этом случае, мы можем использовать мультипликативную функцию для расчета максимального потока.

Таким образом, отличие между аддитивной и мультипликативной функциями в динамическом программировании заключается в том, как они используются для решения конкретных задач. Аддитивная функция используется в задачах на кратчайший путь, когда затраты на переход не зависят от других переходов, а мультипликативная функция используется в задачах на максимальный поток и минимальный разрез, когда затраты на переход зависят от других переходов.

* + - 1. **Каков принцип оптимальности Беллмана?**

Принцип оптимальности Беллмана является основополагающим принципом в теории динамического программирования. Он заключается в том, что для того чтобы найти оптимальное решение задачи, достаточно знать оптимальные решения всех ее подзадач. Другими словами, если задача может быть разбита на более мелкие подзадачи, то оптимальное решение задачи может быть получено путем комбинирования оптимальных решений этих подзадач.

Математически принцип оптимальности Беллмана формулируется следующим образом: если x\* является оптимальным решением задачи, то для любого состояния i в задаче должно выполняться соотношение:

V(i) = min { f(i, j) + V(j) },

где V(i) - значение оптимальной целевой функции, f(i,j) - стоимость перехода из состояния i в состояние j, и минимум берется по всем состояниям j, доступным из i.

Иными словами, оптимальное решение задачи может быть выражено через оптимальные решения ее подзадач. Это позволяет использовать рекурсивные алгоритмы и динамическое программирование для эффективного решения сложных задач.

* + - 1. **Что такое рекурсивный алгоритм?**

Рекурсивный алгоритм - это алгоритм, который использует самого себя для решения задачи. Он базируется на идее разбиения большой задачи на более мелкие подзадачи того же типа. Эти подзадачи решаются вызовом того же алгоритма, что и для исходной задачи, но с меньшими размерами. Рекурсивный алгоритм продолжает рекурсивно вызывать самого себя, пока не достигнет базового случая, который имеет простое и явное решение.

Рекурсивный алгоритм может быть описан с помощью рекурсивной функции, которая вызывает саму себя с другими аргументами, чтобы решить более мелкую задачу. Основная идея рекурсивного алгоритма заключается в том, что он сводит большую задачу к более простым подзадачам, что позволяет эффективно решать сложные задачи.

Примером рекурсивного алгоритма может быть алгоритм вычисления факториала, где для вычисления факториала числа n необходимо вычислить факториалы всех меньших чисел. Другой пример - алгоритм сортировки слиянием, который рекурсивно разбивает список на две половины, сортирует их, а затем объединяет отсортированные половины в один отсортированный список.

* + - 1. **Что такое рекурсивная функция?**

Рекурсивная функция - это функция, которая вызывает сама себя в своем определении. Такие функции используются для решения задач, которые могут быть сведены к более маленьким подзадачам того же типа.

Рекурсивные функции могут иметь базовый случай и рекурсивный случай. Базовый случай - это случай, в котором функция просто возвращает значение, не вызывая саму себя. Рекурсивный случай - это случай, в котором функция вызывает саму себя с другими параметрами, чтобы решить более маленькие версии той же задачи. Это продолжается, пока не будет достигнут базовый случай, когда функция перестает вызывать саму себя и начинает возвращать значения.

Примером рекурсивной функции может быть функция вычисления факториала. Факториал числа n определяется как произведение всех целых чисел от 1 до n. Функция вычисления факториала может быть определена следующим образом:

def factorial(n):

if n == 0:

return 1

else:

return n \* factorial(n-1)

Эта функция имеет базовый случай, когда n равно 0, и рекурсивный случай, когда n больше 0. Если n равно 0, то функция возвращает 1. Если n больше 0, то функция вызывает саму себя с аргументом n-1, чтобы решить более маленькую версию задачи вычисления факториала.

* + - 1. **Что такое системный стек?**

Системный стек (или стек вызовов) - это структура данных, используемая операционной системой для хранения информации о вызовах функций в программе.

Когда функция вызывается в программе, информация о текущем состоянии функции, такой как адрес возврата и локальные переменные, сохраняется в системном стеке. Когда функция завершается, эта информация извлекается из стека и программа продолжает выполнение с места, где она остановилась перед вызовом функции.

Системный стек также используется для управления вложенными вызовами функций. Если функция вызывает другую функцию, информация о первой функции сохраняется в стеке, пока вторая функция не завершится. Затем информация о первой функции извлекается из стека, и первая функция продолжает свое выполнение с места, где она остановилась перед вызовом второй функции.

Системный стек имеет ограниченный размер, поэтому если в программе слишком много вложенных вызовов функций, может произойти переполнение стека, что приведет к аварийному завершению программы.

* + - 1. **Объясните понятие «глубина рекурсии»**

Глубина рекурсии - это количество вложенных вызовов функции, которые происходят во время выполнения рекурсивного алгоритма.

Когда функция вызывает сама себя, это называется рекурсией. Если такие вызовы происходят внутри других вызовов функции, то глубина рекурсии увеличивается. Например, если функция вызывает сама себя два раза в теле функции, глубина рекурсии будет равна двум.

Глубина рекурсии может быть важным фактором при проектировании рекурсивных алгоритмов, потому что каждый вложенный вызов функции занимает определенное количество памяти и времени. Если глубина рекурсии слишком большая, то это может привести к переполнению стека или другим проблемам производительности.

Поэтому при реализации рекурсивных алгоритмов необходимо следить за глубиной рекурсии и при необходимости ограничивать ее, например, используя циклы вместо рекурсивных вызовов функций.

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

Принцип «разделяй и властвуй» - это стратегия решения задач, которая заключается в разбиении большой задачи на более мелкие и более простые подзадачи, решение которых проще, а затем объединении решений подзадач в общее решение исходной задачи.

Эта стратегия обычно используется для задач, которые могут быть разбиты на меньшие, независимые задачи, которые могут быть решены параллельно. Каждая подзадача решается отдельно, а затем ее решение объединяется с решениями других подзадач для получения общего решения.

Например, можно использовать принцип «разделяй и властвуй» для сортировки массива. Для этого массив разбивается на две части, каждая из которых сортируется отдельно. Затем отсортированные части объединяются вместе, чтобы получить отсортированный массив. Этот процесс повторяется для каждой части, пока весь массив не будет отсортирован.

В общем случае, принцип «разделяй и властвуй» помогает снизить сложность задачи путем разбиения ее на более простые подзадачи, которые могут быть решены эффективнее. Однако для применения этого принципа необходимо уметь разбивать задачу на подзадачи и правильно объединять их решения.

* + - 1. **Что такое редакционное расстояние?**

Редакционное расстояние (также известное как расстояние Левенштейна) - это мера разницы между двумя строками в терминах минимального количества операций вставки, удаления и замены символов, необходимых для превращения одной строки в другую.

Каждая из этих операций имеет определенную стоимость, которая может быть задана в соответствии с конкретными требованиями задачи. Например, в задачах автозамены или исправления опечаток стоимость вставки и замены символов может быть больше, чем стоимость удаления символов, так как удаление символов может быть более простым и дешевым способом исправления ошибок.

Редакционное расстояние широко используется в компьютерной лингвистике, а также в других областях, включая биоинформатику, графический дизайн, робототехнику и машинное обучение.

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

Подпоследовательность - это последовательность элементов, которые получаются из исходной последовательности путем удаления некоторых элементов, возможно, не подряд.

Например, пусть у нас есть последовательность [1, 2, 3, 4, 5, 6, 7, 8], и мы хотим получить подпоследовательность, состоящую только из четных чисел. Мы можем удалить элементы 1, 3, 5 и 7, чтобы получить подпоследовательность [2, 4, 6, 8].

Существует несколько подходов к поиску подпоследовательностей, в зависимости от задачи. Некоторые из них включают в себя перебор всех возможных подпоследовательностей и проверку каждой на соответствие требованиям, или использование алгоритмов динамического программирования.

Например, один из известных алгоритмов для нахождения наибольшей общей подпоследовательности (LCS) между двумя последовательностями - это алгоритм Дини-Кройгера. Он использует динамическое программирование и работает за время O(mn), где m и n - длины двух исходных последовательностей.