**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc199070382)

[1. ТЕОРЕТИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ И АЛГОРИТМОВ ЕЕ РЕШЕНИЯ 5](#_Toc199070383)

[1.1 Формальное описание выбранной задачи 5](#_Toc199070384)

[1.2 ИССЛЕДОВАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ 6](#_Toc199070385)

[1.2.1 Наивный алгоритм 6](#_Toc199070386)

[1.2.2 Алгоритм Кнута-Мориса-Пратта(КМП) 7](#_Toc199070387)

[2 РЕАЛИЗАЦИЯ ИССЛЕДУЕМОГО АЛГОРИТМА 12](#_Toc199070388)

[2.1 Программная реализация 12](#_Toc199070389)

[2.2 Unit тестирование написанного кода 19](#_Toc199070390)

[3 ИНТЕГРАЦИЯ В ВЕБ-ИНТЕРФЕЙС 23](#_Toc199070391)

[3.1 Постановка задачи прикладного применения исследуемого алгоритма 23](#_Toc199070392)

[3.2 Выбор технологического стека 25](#_Toc199070393)

[3.3 Реализация фронт и бэк частей 26](#_Toc199070394)

[ЗАКЛЮЧЕНИЕ 28](#_Toc199070395)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 29](#_Toc199070396)

[ПРИЛОЖЕНИЕ А 30](#_Toc199070397)

[ПРИЛОЖЕНИЕ B 35](#_Toc199070398)

[ПРИЛОЖЕНИЕ C 41](#_Toc199070399)

[ПРИЛОЖЕНИЕ D 45](#_Toc199070400)

# **ВВЕДЕНИЕ**

**Актуальность**: поиск подстроки является одной из ключевых задач в компьютерных науках, находящий применение в обработке текстов, системах антиплагиата и других областях. Эффективность алгоритмов поиска подстроки, зачастую, определяет производительность таких систем.

**Цель работы**: разработать программную реализацию алгоритма Кнута-Морриса-Пратта(КМП) для поиска подстроки и интегрировать его в веб-сервис для практического использования.

**Объект исследования**: алгоритм Кнута-Морриса-Пратт, предназначенный для поиска подстроки в тексте.

Предмет исследования: реализация префикс-функции, которая лежит в основе алгоритма КМП

**Задачи**:

1. Провести теоретический анализ задачи поиска подстроки и методов ее решения.
2. Сравнить точность и скорость работы КМП и других алгоритмов поиска подстроки.
3. Реализовать алгоритм КМП на языках программирования Python и C++ с подробными комментариями.
4. Выполнить юнит-тестирование алгоритма.
5. Создать веб-интерфейс на JavaScript для взаимодействия с пользователем

**Практическая значимость**: Результаты исследования могут быть использованы в системах антиплагиата для быстрого и точного выявления заимствований в текстах.

**Методы исследования:**

* Теоретические: анализ научной литературы, изучение алгоритмов сортировки, сравнение их временной и пространственной сложности.
* Практические: программная реализация КМП, написание юнит-тестов (позитивных, негативных, граничных), разработка веб-приложения с использованием Spring.

**Структура работы:** Работа состоит из трёх глав: теоретический анализ задачи и алгоритмов её решения, реализация исследуемого алгоритма и интеграция в веб-сервис.

# **ТЕОРЕТИЧЕСКИЙ АНАЛИЗ ЗАДАЧИ И АЛГОРИТМОВ ЕЕ РЕШЕНИЯ**

## Формальное описание выбранной задачи

**Наименование**: задача поиска подстроки  
Описание задачи: Задача поиска подстроки заключается в обнаружении всех вхождений заданного образца (паттерна) в тексте. Формально: для строки T (длины n) и образца P (длины m) требуется найти все индексы i, где подстрока T[i..i+m-1] совпадает с P.

**Математическая модель**

Дано две строки: текст (T) длиной (n) и подстрока (P) длиной (m), при этом 1 **≤** m **≤** n

**Цель**: найти все индексы (i), начиная с которых подстрока (P) содержится в тексте(T), если такого индекса нет – вернуть пустое множество

**Целевая функция**: минимизация времени исполнения алгоритма, выраженная через асимптотическую сложность.

**Ограничения**:

1. Строки T и P состоят из некоторого конечного алфавита.
2. n и m – положительные целые числа, причем n ≥ m
3. Шаблон P может встречаться в тексте T ноль и более раз.

## ИССЛЕДОВАНИЕ АЛГОРИТМОВ РЕШЕНИЯ ЗАДАЧИ

Рассмотрим дав алгоритма решения задачи поиска подстроки, напишем реализацию(псевдокод), разберем принцип работы.

### **1.2.1 Наивный алгоритм**

Идея данного алгоритма заключается в проверке каждой возможной позиции i в тексте T и сравнении символов с подстрокой P, если совпадение найдено, то индекс записываем результат.

**Принцип работы**: наивный алгоритм последовательно проверяет каждую возможную позицию i в тексте T. Для каждой позиции символы T[i..i+m-1] сравниваются с P[0..m-1]. При несовпадении проверка прерывается, и алгоритм переходит к следующей позиции i+1.

С псевдокодом данного алгоритма можно ознакомиться на Рисунке 1

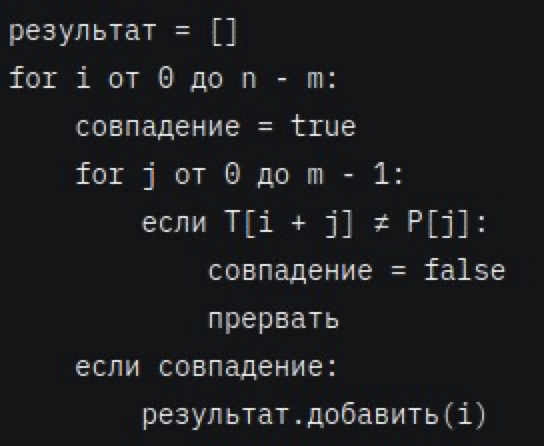


Рисунок 1 – Псевдокод линейного алгоритма

В лучшем случае асимптотическая сложность данного алгоритма составляет O(n), например, если шаблон не найден или сразу найден. Худшая O(n\*m) при множестве частичных совпадений.

Наивный алгоритм прост в реализации, но неэффективен для больших текстов и шаблонов из-за большого числа повторных сравнений.

### **1.2.2 Алгоритм Кнута-Мориса-Пратта(КМП)**

КМП избегает повторных сравнений символов текста за счет использования **префикс-функции**, которая хранит информацию о совпадающих префиксах и суффиксах образца. Это позволяет выполнять оптимальные сдвиги при несовпадениях.

Алгоритм использует два этапа:

1. Построение префикс-функции (массива pi) для подстроки.
2. Поиск подстрок в тексте с использованием информации из pi.

Для строки P префикс-функция pi (определяется как массив, где pi[i] — длина наибольшего собственного префикса подстроки P[0..i], совпадающего с её суффиксом.

Алгоритм построения pi:

1. Проверка крайних случаев
2. Декларация массива pi и инициализация первого значения pi[0].
3. Использование двух указателей: k (длина текущего совпадающего префикса) и i (индекс для обхода образца).
4. Если не совпадают, уменьшаем pi до pi[pi-1] (возвращаемся к предыдущему совпадению).
5. Если символы P[i] и P[k] совпадают, увеличиваем k и в конце обхода записываем его в pi[i].

Сложность алгоритма нахождения префикс-функции составляет O(m), где m – длина подстроки.

С реализацией данного алгоритма на языке программирования Java можно ознакомиться на Рисунке 2.

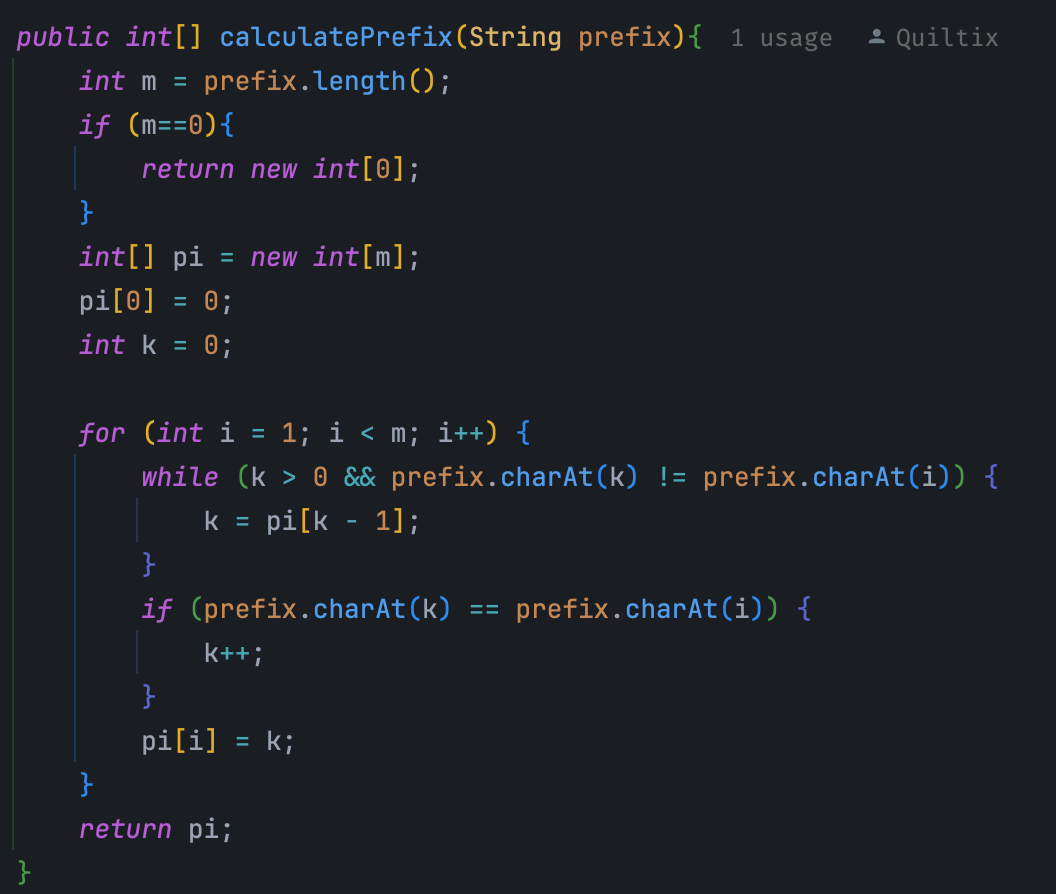


Рисунок 2 – Нахождение префикс функции на языке программирования Java

Следующим этапом является поиск всех вхождений подстроки в строку с использованием префикс-функции

Шаги выполнения алгоритма:

1. Предобработка, которая включает вызов метода для расчета pi
2. Инициализация необходимых переменных и обработка крайних случаев
3. Прохождение по элементам текста и их сравнение с элементами подстроки
   1. В случае совпадения элементов, сдвигаем указатели и снова сравниваем, если таким образом все элементы подстроки совпадают с подряд идущими элементами строки, то записываем индекс начала и сдвигаем указатель подстроки на число, которое записано в предыдущей ячейки префикс-функции (j=p[j-1])
   2. Если элементы не совпадают, то оптимально сдвигаем указатель подстроки на на число, которое записано в предыдущей ячейки префикс-функции (j=p[j-1])
   3. Если совпадений не будет найдено, результатом работы метода станет пустой список

Благодаря такому алгоритму можно избежать избыточных сравнений.

С реализацией данного алгоритма можно ознакомиться на Рисунке 3

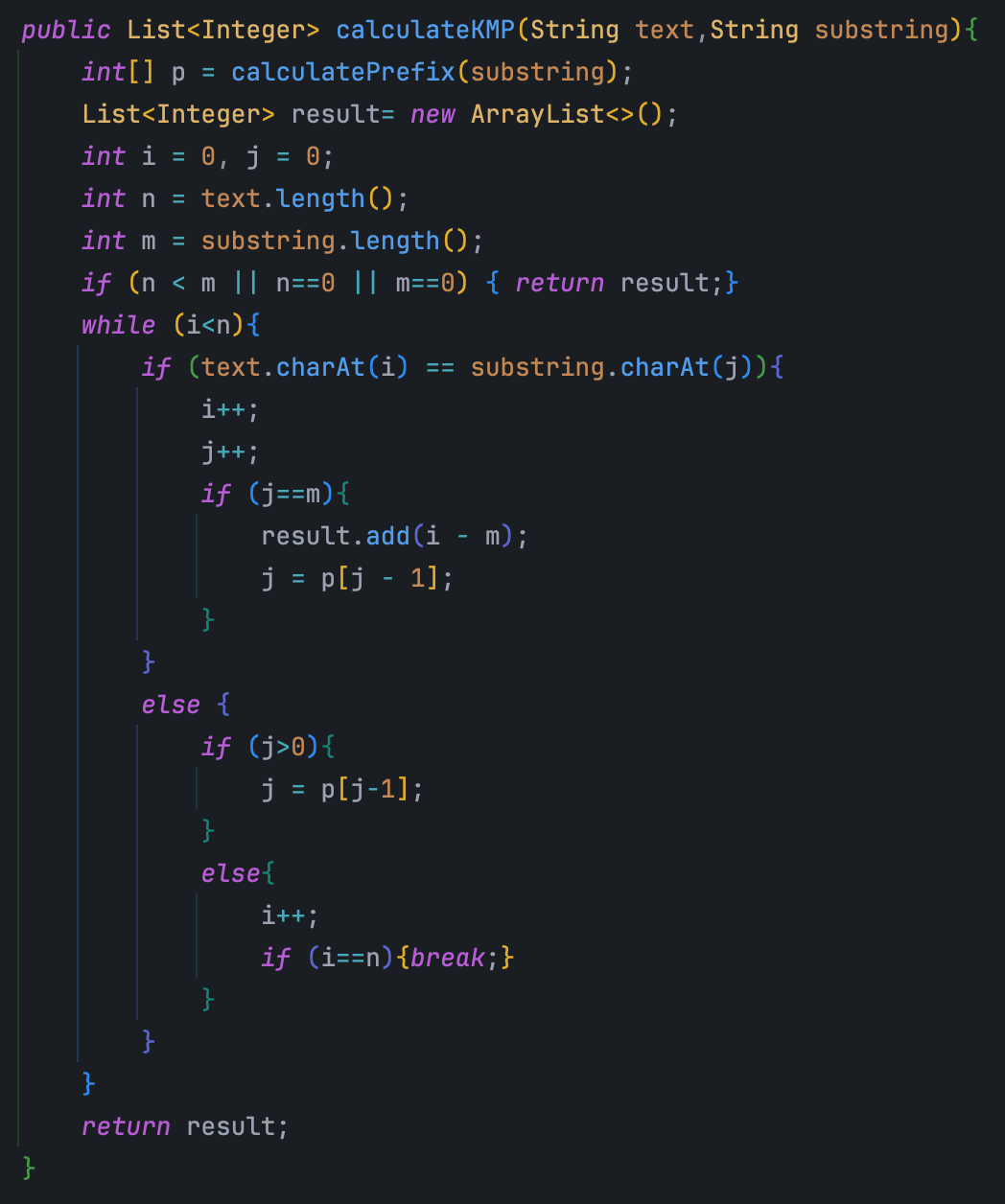


Рисунок 3 – Поиск всех вхождений подстроки в строку с использованием префикс-функции на языке программирования Java

Таким образом асимптотическая сложность данного алгоритма - O(n + m). Префикс-функция строится за O(m), поиск выполняется — за O(n). Пространственная сложность – O(m) для хранения массива pi.

КМП гарантирует линейное время выполнения, что делает его идеальным для задач с большими текстами, таких как антиплагиат. Префикс-функция минимизирует повторные сравнения, используя информацию о структуре шаблона.

В результате разбора двух алгоритмов, можно сказать, что Алгоритм КМП превосходит наивный подход за счет использования префикс-функции, что исключает избыточные сравнения. Его асимптотическая сложность O(n + m) делает его предпочтительным для поиска точных совпадений фрагментов текста. Префикс-функция позволяет эффективно пропускать уже проверенные символы, минимизируя количество сравнений. Наивный алгоритм может проще в реализации, но его скорость работы, зачастую, делает его неэффективным для больших текстов и шаблонов из-за большого количества повторений.

# **2 РЕАЛИЗАЦИЯ ИССЛЕДУЕМОГО АЛГОРИТМА**

### **2.1 Программная реализация**

Реализуем алгоритм КМП на Java и C++.

Реализация алгоритма КМП на Java состоит из двух методов. Метод calculatePrefix вычисляет префикс-функцию, которая определяет, насколько можно сместить шаблон при несоответствии. Метод calculateKMP использует эту функцию для поиска всех вхождений шаблона в тексте, возвращая список индексов. Код эффективен и подходит для обработки больших текстов, например, в системах антиплагиата.

Вход: Строка prefix — шаблон, для которого вычисляется префикс-функция.

Выход: Массив целых чисел pi, где pi[i] — длина самого длинного префикса-суффикса для подстроки prefix[0..i].

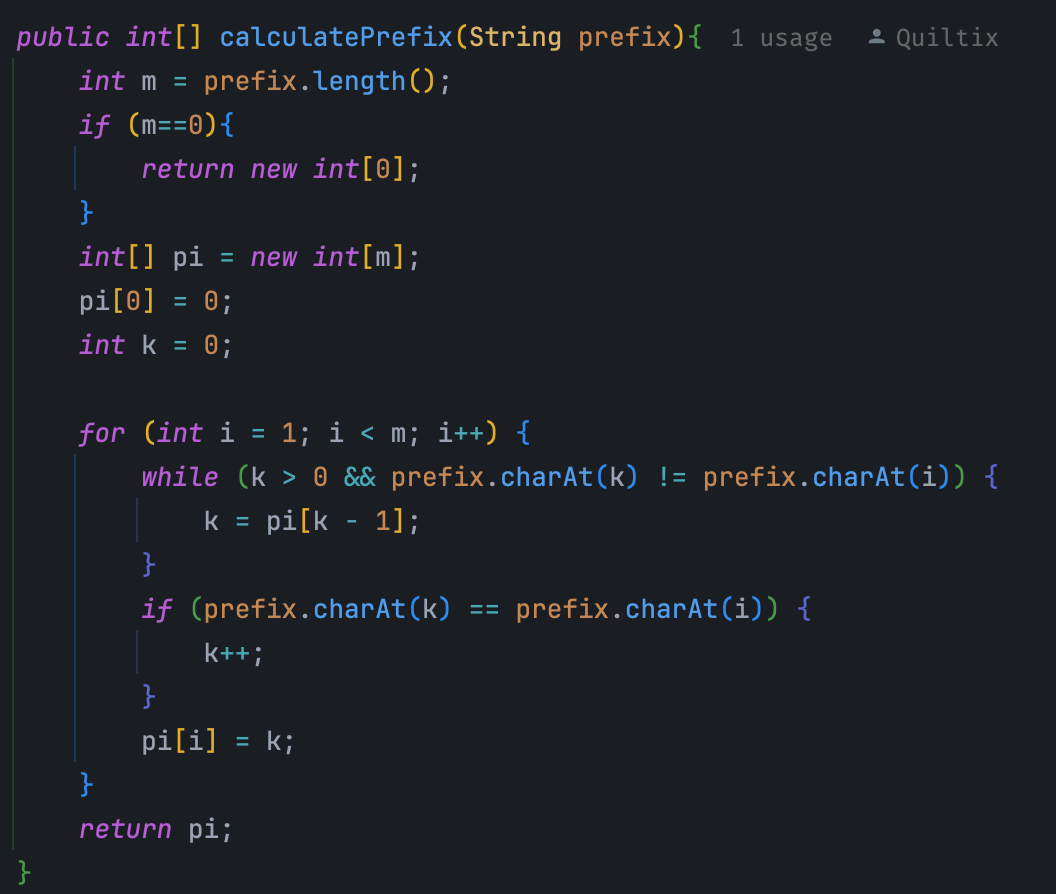


Рисунок 4 – Реализация нахождения префикс-функции на языке программирования Java

Метод calculatePrefix (Рисунок 4) вычисляет префикс-функцию (массив pi)для заданного шаблона. Префикс-функция определяет длину самого длинного собственного префикса подстроки prefix[0..i], который одновременно является суффиксом этой подстроки. Эта информация позволяет алгоритму КМП эффективно смещать шаблон при несоответствии, избегая ненужных сравнений.

**Логика работы:**

1. **Обработка краевого случая:** если длина шаблона равна нулю (m = 0), метод возвращает пустой массив, так как префикс-функция не имеет смысла для пустой строки.
2. **Инициализация:** создается массив pi размером m, устанавливается pi[0]=0, поскольку для первого символа нет собственных префиксов и вводится переменная k, инициализируемая нулем, которая отслеживает длину текущего префикса-суффикса.
3. **Основной цикл:** Если символы prefix[k] и prefix[i] не совпадают и k > 0, уменьшаем k до значения pi[k-1]. Это позволяет "откатиться" к предыдущему возможному префиксу-суффиксу. Если символы совпадают, увеличиваем k. bЗаписываем pi[i] = k, фиксируя длину префикса-суффикса для текущей позиции.
4. После обработки всех позиций возвращается массив pi.

Метод calculateKMP(Рисунок 5) выполняет поиск всех вхождений шаблона в тексте, используя префикс-функцию для оптимизации процесса. Он возвращает список индексов, с которых начинаются совпадения шаблона в тексте.

Вход: Строки text (текст для поиска) и substring (шаблон).

Выход: Список List<Integer>, содержащий индексы начала каждого вхождения шаблона в тексте.

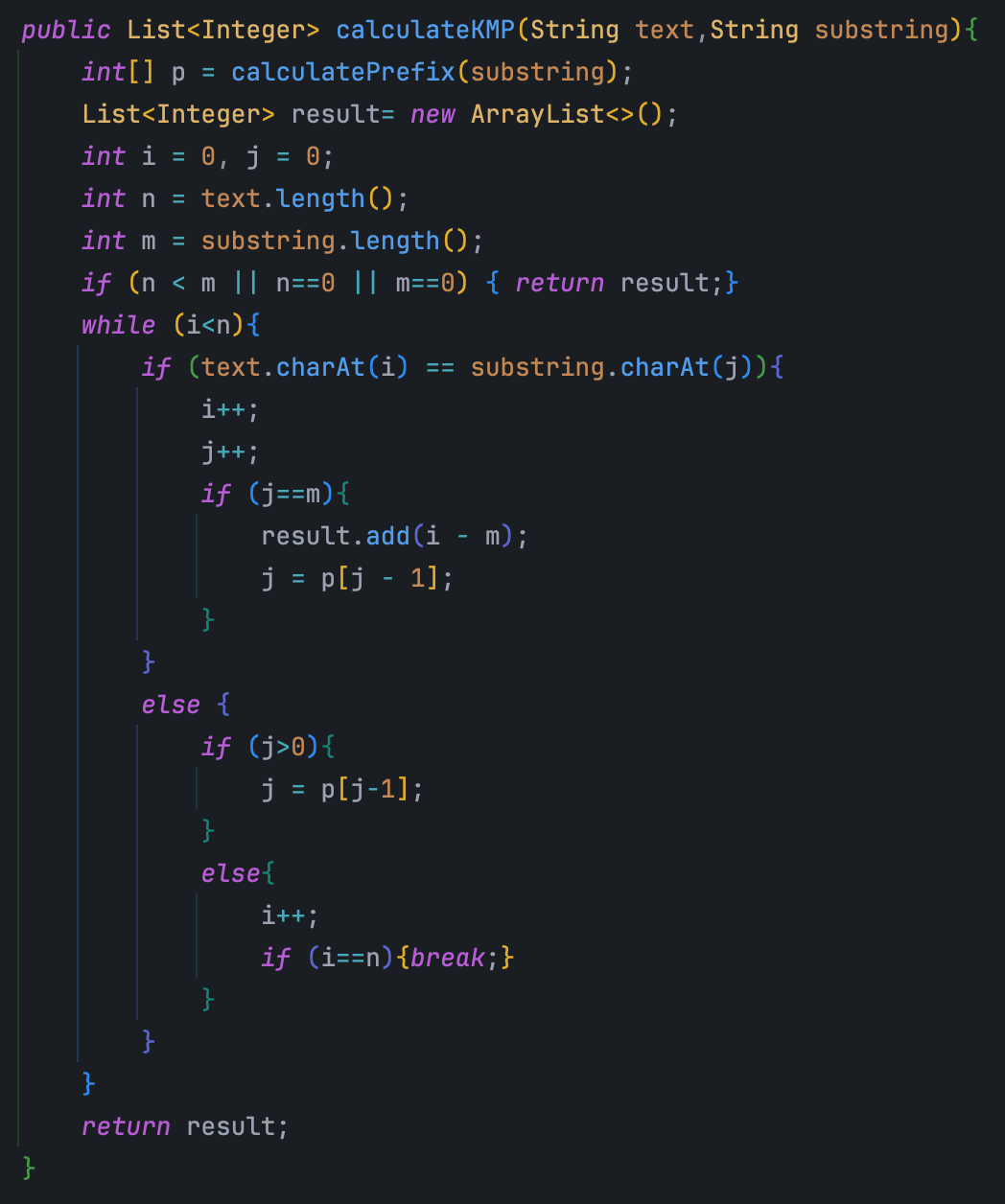


Рисунок 5 – Реализация метода calculateKMP на языке программирования Java

**Логика работы:**

1. **Предварительная обработка:** 
   * Вычисляется префикс-функция для шаблона с помощью метода calculatePrefix, результат сохраняется в массиве p.
   * Инициализируется пустой список result для хранения индексов вхождений.
   * Вводятся указатели:
     + i — текущая позиция в тексте (начинает с 0).
     + j — текущая позиция в шаблоне (начинает с 0).
   * Проверяются краевые случаи:
     + Если длина текста n меньше длины шаблона m, возвращается пустой список.
     + Если текст или шаблон пусты, возвращается пустой список.
2. **Основной цикл поиска:**  
   Пока i < n:
   * Если символы text[i] и substring[j] совпадают:
     + Увеличиваются оба указателя
     + Если j достигает длины шаблона m, найдено вхождение:
       - В список result добавляется индекс начала вхождения i - m Указатель j сбрасывается на основе префикс-функции: j=p[j−1], чтобы продолжить поиск следующего вхождения.
   * Если символы не совпадают:
     + Если j>0, смещаем j на основе префикс-функции: j=p[j−1].
     + Если j=0, увеличиваем i, чтобы перейти к следующему символу текста.
   * Если i достигает, цикл завершается.
3. **Возврат результата:**  
   Возвращается список result, содержащий все индексы вхождений.

Реализация на C++ состоит из двух функций, аналогичных Java-версии: calculatePrefix вычисляет префикс-функцию для шаблона, а calculateKMP выполняет поиск подстроки в тексте.

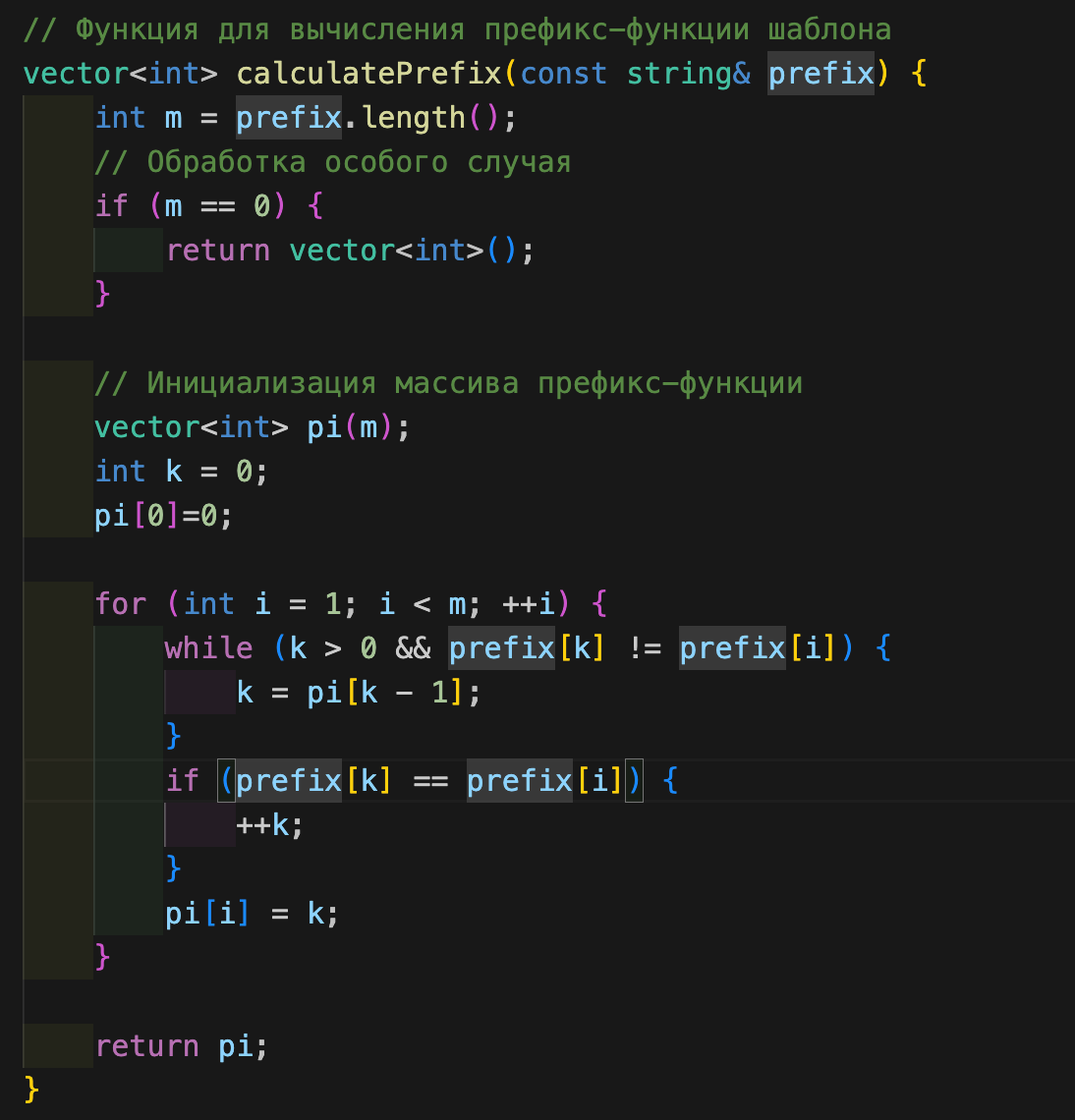


Рисунок 6 – Реализация нахождения префикс-функции на языке программирования C++

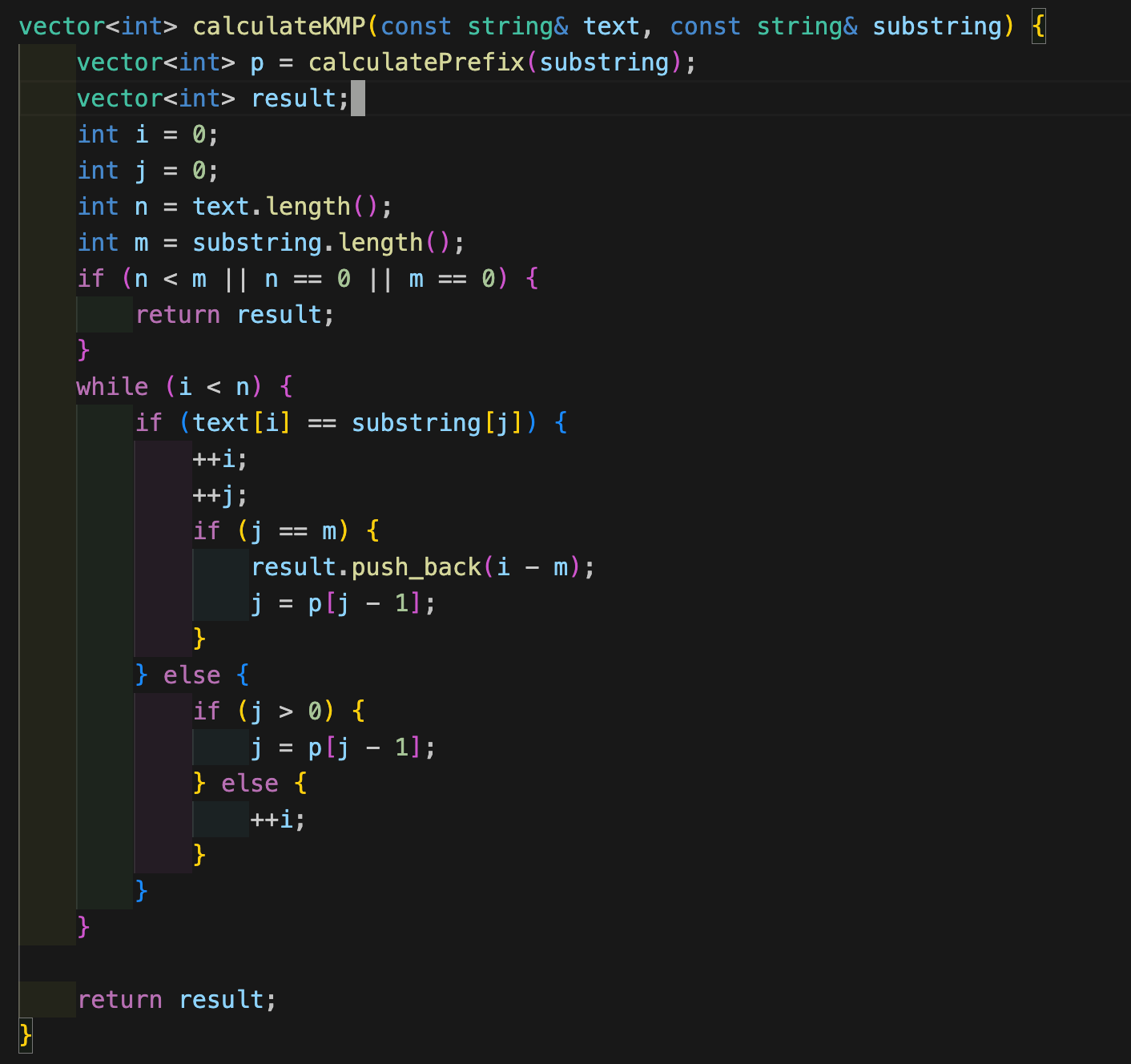


Рисунок 7 – Реализация метода calculateKMP на языке программирования C++

Как можно заметить, реализация алгоритма КМП на C++ аналогична, написанному на Java решению, за исключением использования в C++ такой структуры данных как vector.

### **2.2 Unit тестирование написанного кода**

Юнит-тесты проверяют функциональность методов calculatePrefix и calculateKMP класса AlgorithmService. Тесты разделены на категории: позитивные (корректные входные данные с ожидаемыми результатами), негативные (некорректные или отсутствующие данные) и граничные (краевые случаи, такие как пустые строки или строки минимальной длины). Каждый тест является независимым методом, что упрощает диагностику ошибок и соответствует лучшим практикам тестирования. Тесты используют библиотеку JUnit 5 и основаны на написанном Java коде.

Для тестирования написан следующий код:

public class AlgorithmServiceTest {

private AlgorithmService service;

private UserRepository userRepository;

private AlgorithmRepository algorithmRepository;

@BeforeEach

public void setUp() {

userRepository = Mockito.mock(UserRepository.class);

algorithmRepository = Mockito.mock(AlgorithmRepository.class);

service = new AlgorithmService(userRepository,algorithmRepository);

}

// Позитивные тесты для calculatePrefix

@Test

public void testCalculatePrefix1() {

assertArrayEquals(new int[]{0}, service.calculatePrefix("a"));

}

@Test

public void testCalculatePrefix2() {

assertArrayEquals(new int[]{0, 1}, service.calculatePrefix("aa"));

}

@Test

public void testCalculatePrefix3() {

assertArrayEquals(new int[]{0, 0, 0}, service.calculatePrefix("abc") );

}

@Test

public void testCalculatePrefix4() {

assertArrayEquals(new int[]{0, 0, 1, 2}, service.calculatePrefix("abab"));

}

// Негативный тест для calculatePrefix

@Test

public void testCalculatePrefixEmptyString() {

assertArrayEquals(new int[]{}, service.calculatePrefix(""));

}

// Позитивные тесты для calculateKMP

@Test

public void testCalculateKMP1() {

assertEquals(List.of(0, 2, 4), service.calculateKMP("abababab", "abab"));

}

@Test

public void testCalculateKMP2() {

assertEquals(List.of(0, 3, 6), service.calculateKMP("abcabcabc", "abc"));

}

@Test

public void testCalculateKMP3() {

assertEquals(List.of(1), service.calculateKMP("a ", " "));

}

// Негативные тесты для calculateKMP

@Test

public void testCalculateKMP4() {

assertEquals(List.of(), service.calculateKMP("abc", "d"));

}

@Test

public void testCalculateKMP5() {

assertEquals(List.of(), service.calculateKMP("abc", "ddddd"));

}

// Граничные тесты для calculateKMP

@Test

public void testCalculateKMP6() {

assertEquals(List.of(), service.calculateKMP("", "a"));

}

@Test

public void testCalculateKMP7() {

assertEquals(List.of(), service.calculateKMP("a", ""));

}

}

Юнит-тесты для AlgorithmService обеспечивают полное покрытие функциональности алгоритма КМП, проверяя методы calculatePrefix и calculateKMP на различных сценариях. Позитивные тесты подтверждают корректность работы, негативные — устойчивость к ошибкам, а граничные — обработку краевых случаев. Тесты гарантируют надежность алгоритма для реальных задач, где точность и производительность критичны.

На рисунке 8 можно ознакомиться с результатами работы тестов.

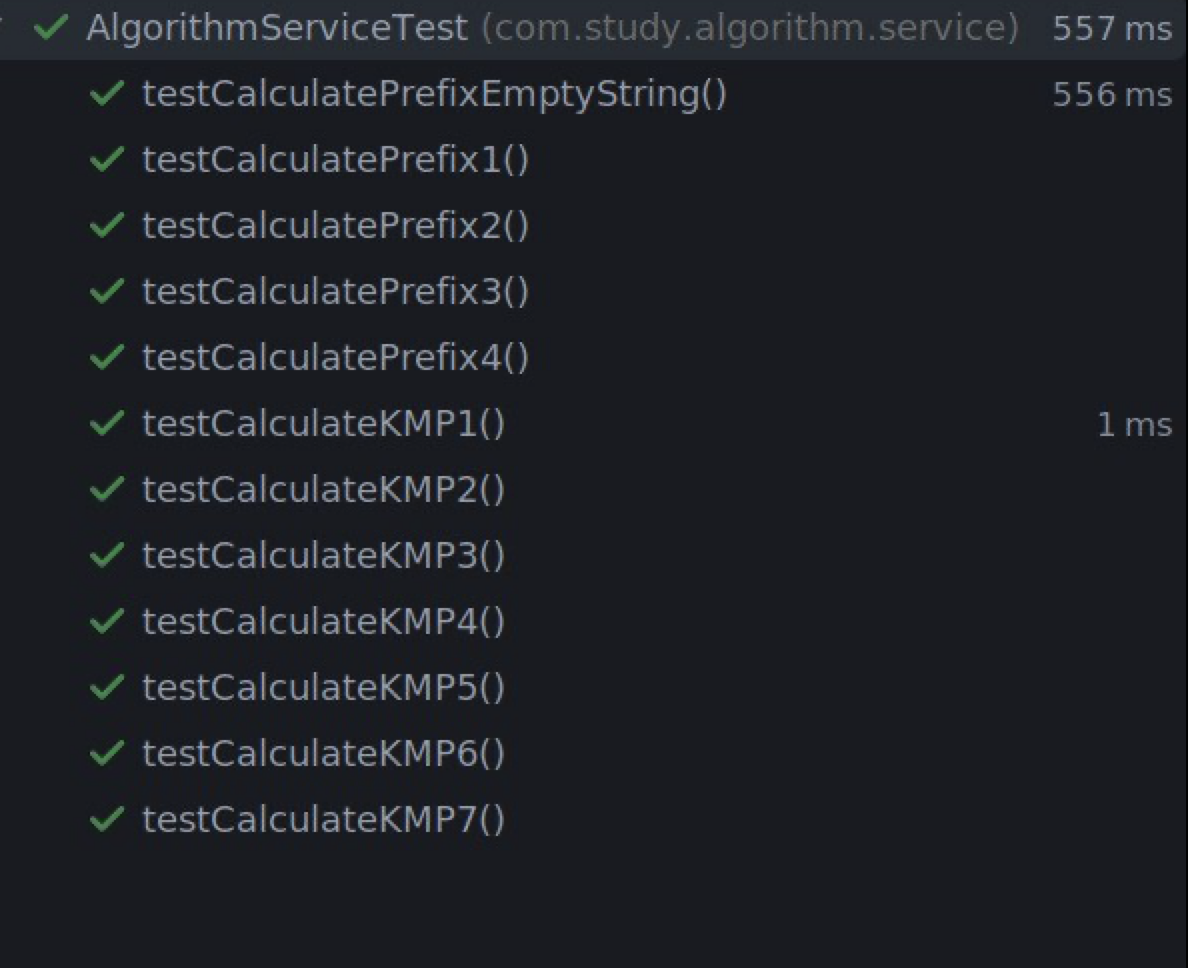


Рисунок 8 – Результат работы тестов

Юнит-тесты для AlgorithmService обеспечивают полное покрытие функциональности алгоритма КМП, проверяя методы calculatePrefix и calculateKMP на различных сценариях. Позитивные тесты подтверждают корректность работы, негативные — устойчивость к ошибкам, а граничные — обработку краевых случаев. Тесты гарантируют надежность алгоритма для реальных задач, где точность и производительность критичны.

# **3 ИНТЕГРАЦИЯ В ВЕБ-ИНТЕРФЕЙС**

### **3.1 Постановка задачи прикладного применения исследуемого алгоритма**

Создать веб-сервис для проверки текстов на наличие заимствований (антиплагиат) с использованием алгоритма Кнута-Морриса-Пратта (КМП). Сервис должен позволять пользователям загружать текст и шаблон. Это делает сервис полезным для проверки текстов на плагиат в образовательных учреждениях.

**Описание задачи:**

* **Входные данные:** Пользователь предоставляет текст (например, статью или эссе) и шаблон (фрагмент, который нужно найти).
* **Обработка:** Алгоритм КМП используется для поиска всех вхождений шаблона в тексте. Результаты (индексы совпадений) возвращаются пользователю.
* **Выходные данные:** Список индексов, где шаблон встречается в тексте, с возможностью визуализации совпадений (например, выделение найденных фрагментов).
* **Применение:** Сервис может использоваться в образовательных учреждениях для проверки студенческих работ на плагиат

### **3.2 Выбор технологического стека**

Для реализации веб-сервиса выбран следующий технологический стек, соответствующий современным тенденциям (Spring, Hibernate, Spring Security, HTML/CSS/JS):

**Бэкенд:**

* + **Spring Boot (Java):** Фреймворк для создания REST API, упрощающий настройку и развертывание. Spring Boot обеспечивает встроенный сервер (Tomcat), управление зависимостями и поддержку REST-контроллеров.
  + **Hibernate (JPA):** ORM-фреймворк для работы с базой данных. Используется для хранения шаблонов и пользовательских данных (логины, роли).
  + **Spring Security:** Обеспечивает авторизацию и аутентификацию пользователей. Используется для ограничения доступа к API только зарегистрированным пользователям.
  + **PostgreSQL:** Мощная база данных для разработки и тестирования обеспечивает расширяемость.
  + **Maven:** Система сборки для управления зависимостями и компиляции проекта.

**Фронтенд:**

* + **HTML5:** Основа структуры веб-страниц.
  + **CSS3:** Стилизация интерфейса, включая адаптивный дизайн для удобства использования на разных устройствах.
  + **JavaScript (с Fetch API):** Обработка пользовательских действий и асинхронных запросов к бэкенду. Используется для динамического обновления страницы без перезагрузки.
* **Дополнительно:**
  + **REST API:** Для взаимодействия между фронтендом и бэкендом. API принимает текст и шаблон, возвращает результаты поиска.
  + **JSON:** Формат обмена данными между фронтендом и бэкендом.

### **3.3 Реализация фронт и бэк частей**

Бэкенд реализован на Spring Webt с использованием Hibernate, Spring Security и JWT-аутентификации. Он предоставляет REST API для выполнения поиска подстроки, сохранения результатов и получения истории поисков. Основные компоненты:

**Модель данных:**

* + User: Сущность для хранения данных пользователей (идентификатор, имя пользователя, пароль).
  + Algorithm: Сущность для хранения поисковых запросов (текст, шаблон, результаты, владелец).

**Сервисы:**

* + AlgorithmService: Содержит логику КМП (calculatePrefix, calculateKMP), выполняет поиск, сохраняет результаты и предоставляет историю поисков.
  + UserService: Управляет регистрацией и аутентификацией пользователей, генерирует JWT-токены.

**Контроллеры:**

* + AlgorithmController: Обрабатывает запросы на поиск и получение истории.
  + AuthController: Обрабатывает регистрацию и вход пользователей.

**Репозитории:**

* + UserRepository: Для работы с пользователями.
  + AlgorithmRepository: Для работы с поисковыми запросами.

**Безопасность:** Spring Security с JWT защищает эндпоинты, требуя токен в заголовке Authorization.

**API-эндпоинты:**

* **POST /api/auth/register**: Регистрация пользователя (принимает RegisterRequestDTO).
* **POST /api/auth/login**: Аутентификация, возвращает JWT-токен (LoginRequestDTO).
* **POST /api/algorithm**: Выполняет поиск КМП, сохраняет результат, возвращает индексы
* **GET /api/algorithm**: Возвращает историю поисков текущего пользователя.
* **GET /api/algorithm/all**: Возвращает все поиски в системе

Фронтенд реализован на HTML, CSS и JavaScript. Он предоставляет интерфейс для регистрации, входа, ввода текста и шаблона, выполнения поиска и просмотра истории.

HTML и JavaScript включают:

* Форму ввода текста и шаблона.
* Кнопки для регистрации и входа
* Секции для отображения истории пользователя и общей истории.
* JavaScript-логику для отправки запросов к бэкенду и обработки ответов.

# **ЗАКЛЮЧЕНИЕ**

В рамках курсовой работы был реализован алгоритм Кнута-Морриса-Пратта для эффективного поиска подстроки в тексте. Алгоритм запрограммирован на Java и C++, протестирован с помощью юнит-тестов и интегрирован в веб-сервис. Веб-сервис, построенный на Spring Boot, позволяет пользователям регистрироваться, входить в систему, вводить текст и шаблон, выполнять поиск и просматривать историю своих запросов. Интеграция КМП в веб-сервис демонстрирует его практическую ценность для систем антиплагиата в образовательных учереждениях. Алгоритм эффективно находит точные совпадения фрагментов текста, что позволяет выявлять заимствования в академических работах или других документах. Линейная сложность обеспечивает быструю обработку больших текстов, а веб-интерфейс делает сервис доступным для преподавателей, студентов и администраторов образовательных платформ. Сохранение результатов в базе данных и возможность просмотра истории повышают удобство использования.

В перспективе возможно улучшение пользовательского интерфейса, внедрение многопоточности или асинхронности и реализация веб сервиса, как полноценного микросервиса.

# **СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ**

* Обзор REST. URL: <https://javarush.com/groups/posts/2486-obzor-rest-chastjh-1-chto-takoe-rest>
* "Алгоритм Кнута – Морриса – Пратта. URL: <https://ru.wikipedia.org/wiki/Алгоритм_Кнута_—_Морриса_—_Пратта>
* “Это маленькое чудо — алгоритм Кнута-Морриса-Пратта (КМП)”. URL: <https://habr.com/ru/articles/307220/>
* Англоязычная документация Spring Framework. URL: [https://www.baeldung.com](https://www.baeldung.com/)
* “Junit part 1”. URL: <https://javarush.com/groups/posts/605-junit>
* “ Swagger (OpenAPI 3.0)” URL: <https://habr.com/ru/articles/541592/>

# ПРИЛОЖЕНИЕ А

Листинг – Котроллеры Java Spring приложения

@RestController

@RequestMapping("/api/algorithm")

public class AlgorithmController {

private final AlgorithmService algorithmService;

public AlgorithmController(AlgorithmService algorithmService) {

this.algorithmService = algorithmService;

}

@PostMapping

@Operation(

summary = "Решить алгоритм KMP",

description = "Вычисляет вхождения образца в тексте с использованием алгоритма Кнута-Морриса-Пратта",

responses = {

@ApiResponse(

responseCode = "200",

description = "Успешное выполнение",

content = @Content(schema = @Schema(implementation = List.class))

),

@ApiResponse(

responseCode = "401",

description = "Пользователь не авторизован"

)

}

)

public ResponseEntity<List<Integer>> solveKMP(Authentication authentication,

@RequestBody RequestAlgorithmDTO requestAlgorithmDTO) {

List<Integer> result = algorithmService.solveKMP(authentication, requestAlgorithmDTO);

return ResponseEntity.ok(result);

}

@GetMapping

@Operation(

summary = "Получить решения пользователя",

description = "Возвращает список всех решений текущего пользователя",

responses = {

@ApiResponse(

responseCode = "200",

description = "Успешное получение данных",

content = @Content(schema = @Schema(implementation = Algorithm.class))

),

@ApiResponse(

responseCode = "401",

description = "Пользователь не авторизован"

)

}

)

public ResponseEntity<List<Algorithm>> getAllSolvesByUser(Authentication authentication) {

List<Algorithm> result = algorithmService.allSolvesByUser(authentication);

return ResponseEntity.ok(result);

}

@GetMapping("/all")

@Operation(

summary = "Получить все решения",

description = "Возвращает список всех решений в системе (требуются права администратора)",

responses = {

@ApiResponse(

responseCode = "200",

description = "Успешное получение данных",

content = @Content(schema = @Schema(implementation = Algorithm.class)))

}

)

public ResponseEntity<List<Algorithm>> getAllSolves() {

List<Algorithm> result = algorithmService.allSolves();

return ResponseEntity.ok(result);

}

}

@RestController

@RequestMapping("/api/auth")

public class AuthController {

private final UserService userService;

public AuthController(UserService userService) {

this.userService = userService;

}

@PostMapping("/register")

@Operation(

summary = "Регистрация пользователя",

description = "Создает нового пользователя в системе",

responses = {

@ApiResponse(

responseCode = "200",

description = "Успешная регистрация",

content = @Content(schema = @Schema(implementation = MessageDTO.class))

),

@ApiResponse(

responseCode = "400",

description = "Некорректные входные данные"

)

}

)

public ResponseEntity <MessageDTO> registerUser (@RequestBody @Valid RegisterRequestDTO registerRequestDTO){

String message = userService.registerUser(registerRequestDTO);

return ResponseEntity.ok().body(new MessageDTO(message));

}

@PostMapping("/login")

@Operation(

summary = "Аутентификация пользователя",

description = "Возвращает JWT токен для авторизованного доступа",

responses = {

@ApiResponse(

responseCode = "200",

description = "Успешная аутентификация",

content = @Content(schema = @Schema(implementation = JwtAuthenticationResponseDTO.class))

),

@ApiResponse(

responseCode = "401",

description = "Неверные учетные данные"

)

}

)

public ResponseEntity<JwtAuthenticationResponseDTO> authenticateUser(@Valid @RequestBody LoginRequestDTO loginRequestDTO) {

String token = userService.authenticateUser(loginRequestDTO);

return ResponseEntity.ok().body(new JwtAuthenticationResponseDTO(token));

}

}  
@Controller

public class WebController {

@GetMapping("/")

public String index() {

return "index.html";

}

@GetMapping("/login")

public String login() {

return "login.html";

}

}

# ПРИЛОЖЕНИЕ B

Листинг – Сервисы Spring Java приложения.

@Service

@Transactional

public class AlgorithmService {

private final UserRepository userRepository;

private final AlgorithmRepository algorithmRepository;

public AlgorithmService(UserRepository userRepository, AlgorithmRepository algorithmRepository) {

this.userRepository = userRepository;

this.algorithmRepository = algorithmRepository;

}

public List<Integer> solveKMP(Authentication authentication, RequestAlgorithmDTO requestAlgorithmDTO){

String username = authentication.getName();

User user = userRepository.findUserByUsername(username).orElseThrow(

()-> new UsernameNotFoundException("User not found"));

List<Integer> resKMP = calculateKMP(requestAlgorithmDTO.getText(),requestAlgorithmDTO.getSubstring());

Algorithm algorithm = Algorithm.builder()

.text(requestAlgorithmDTO.getText())

.substring(requestAlgorithmDTO.getSubstring())

.result(resKMP)

.owner(user)

.build();

algorithmRepository.save(algorithm);

return resKMP;

}

public List<Algorithm> allSolvesByUser(Authentication authentication) {

String username = authentication.getName();

User user = userRepository.findUserByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found: " + username));

return algorithmRepository.findByOwner(user);

}

public List<Algorithm> allSolves() {

return algorithmRepository.findAll();

}

public int[] calculatePrefix(String prefix){

int m = prefix.length();

if (m==0){

return new int[0];

}

int[] pi = new int[m];

pi[0] = 0;

int k = 0;

for (int i = 1; i < m; i++) {

while (k > 0 && prefix.charAt(k) != prefix.charAt(i)) {

k = pi[k - 1];

}

if (prefix.charAt(k) == prefix.charAt(i)) {

k++;

}

pi[i] = k;

}

return pi;

}

public List<Integer> calculateKMP(String text,String substring){

int[] p = calculatePrefix(substring);

List<Integer> result= new ArrayList<>();

int i = 0, j = 0;

int n = text.length();

int m = substring.length();

if (n < m || n==0 || m==0) { return result;}

while (i<n){

if (text.charAt(i) == substring.charAt(j)){

i++;

j++;

if (j==m){

result.add(i - m);

j = p[j - 1];

}

}

else {

if (j>0){

j = p[j-1];

}

else{

i++;

if (i==n){break;}

}

}

}

return result;

}

}

@Service

public class CustomUserDetailsService implements UserDetailsService {

UserRepository userRepository;

public CustomUserDetailsService(UserRepository userRepository) {

this.userRepository = userRepository;

}

@Override

public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

try {

User user = userRepository.findUserByUsername(username)

.orElseThrow(() -> new UsernameNotFoundException("User not found"));

return new org.springframework.security.core.userdetails.User(

user.getUsername(),

user.getPassword(),

Collections.singleton(new SimpleGrantedAuthority("ROLE\_USER")));

}

catch (UsernameNotFoundException e){

return null;

}

}

}

@Service

public class UserService {

private UserRepository userRepository;

private PasswordEncoder passwordEncoder;

private AuthenticationManager authenticationManager;

private JwtTokenProvider jwtTokenProvider;

public UserService(UserRepository userRepository, PasswordEncoder passwordEncoder, AuthenticationManager authenticationManager, JwtTokenProvider jwtTokenProvider) {

this.userRepository = userRepository;

this.passwordEncoder = passwordEncoder;

this.authenticationManager = authenticationManager;

this.jwtTokenProvider = jwtTokenProvider;

}

@Transactional

public String registerUser(RegisterRequestDTO requestDTO){

if(userRepository.findUserByUsername(requestDTO.getUsername()).isPresent()){

throw new IllegalArgumentException("Username already defined");

}

User user = new User();

user.setUsername(requestDTO.getUsername());

user.setPassword(passwordEncoder.encode(requestDTO.getPassword()));

userRepository.save(user);

return "Successful";

}

public String authenticateUser(LoginRequestDTO loginRequestDTO){

if(userRepository.findUserByUsername(loginRequestDTO.getUsername()).isEmpty()){

throw new UsernameNotFoundException("User not found");

}

UsernamePasswordAuthenticationToken authToken =

new UsernamePasswordAuthenticationToken(loginRequestDTO.getUsername(), loginRequestDTO.getPassword());

Authentication authentication;

try {

authentication = authenticationManager.authenticate(authToken);

} catch (BadCredentialsException ex) {

throw new BadCredentialsException("Invalid password");

}

SecurityContextHolder.getContext().setAuthentication(authentication);

return jwtTokenProvider.generateToken(authentication);

}

}

# ПРИЛОЖЕНИЕ C

Листинг – HTML код приложения

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Авторизация</title>

<link rel="stylesheet" href="/style.css">

</head>

<body>

<div class="auth-container">

<div class="auth-card glassmorphism">

<div class="tabs">

<button class="tab active" data-tab="login">Вход</button>

<button class="tab" data-tab="register">Регистрация</button>

</div>

<form id="loginForm" class="auth-form active">

<div class="input-group">

<label for="loginUsername">Логин</label>

<input type="text" id="loginUsername" required>

</div>

<div class="input-group">

<label for="loginPassword">Пароль</label>

<input type="password" id="loginPassword" required>

</div>

<button type="submit" class="gradient-btn">Войти</button>

<div id="loginError" class="error-message"></div>

</form>

<form id="registerForm" class="auth-form">

<div class="input-group">

<label for="registerUsername">Логин</label>

<input type="text" id="registerUsername" required>

</div>

<div class="input-group">

<label for="registerPassword">Пароль</label>

<input type="password" id="registerPassword" required minlength="6">

</div>

<button type="submit" class="gradient-btn">Зарегистрироваться</button>

<div id="registerError" class="error-message"></div>

</form>

</div>

</div>

<script src="/auth.js"></script>

</body>

</html>

<html lang="ru">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Авторизация</title>

<link rel="stylesheet" href="/style.css">

</head>

<body>

<div class="auth-container">

<div class="auth-card glassmorphism">

<div class="tabs">

<button class="tab active" data-tab="login">Вход</button>

<button class="tab" data-tab="register">Регистрация</button>

</div>

<form id="loginForm" class="auth-form active">

<div class="input-group">

<label for="loginUsername">Логин</label>

<input type="text" id="loginUsername" required>

</div>

<div class="input-group">

<label for="loginPassword">Пароль</label>

<input type="password" id="loginPassword" required>

</div>

<button type="submit" class="gradient-btn">Войти</button>

<div id="loginError" class="error-message"></div>

</form>

<form id="registerForm" class="auth-form">

<div class="input-group">

<label for="registerUsername">Логин</label>

<input type="text" id="registerUsername" required>

</div>

<div class="input-group">

<label for="registerPassword">Пароль</label>

<input type="password" id="registerPassword" required minlength="6">

</div>

<button type="submit" class="gradient-btn">Зарегистрироваться</button>

<div id="registerError" class="error-message"></div>

</form>

</div>

</div>

<script src="/auth.js"></script>

</body>

</html>

# ПРИЛОЖЕНИЕ D

Листинг – JavaScript код проекта

document.addEventListener('DOMContentLoaded', () => {

if (!localStorage.getItem('token')) {

window.location.href = '/login';

return;

}

const logoutBtn = document.getElementById('logout-btn');

const searchForm = document.querySelector('.search-form');

const resultDiv = document.getElementById('result');

const userHistoryDiv = document.getElementById('user-history');

const allHistoryDiv = document.getElementById('all-history');

logoutBtn.addEventListener('click', () => {

localStorage.removeItem('token');

window.location.href = '/login';

});

searchForm.addEventListener('submit', async (e) => {

e.preventDefault();

const text = document.getElementById('string').value.trim();

const substring = document.getElementById('substring').value.trim();

if (!text || !substring) {

showError("Заполните оба поля!");

return;

}

try {

const response = await fetch('http://localhost:8080/api/algorithm', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'Authorization': `Bearer ${localStorage.getItem('token')}`

},

body: JSON.stringify({ text, substring })

});

if (response.status === 401) {

localStorage.removeItem('token');

window.location.href = '/login';

return;

}

const data = await response.json();

showResult(data);

loadHistory();

} catch (error) {

showError(error.message);

}

});

function showError(text) {

resultDiv.innerHTML = `<div class="error">${text}</div>`;

}

function showResult(data) {

const resultHTML = data.length > 0

? `Найденные позиции: ${data.join(', ')}`

: `<div class="error">Подстрока не найдена</div>`;

resultDiv.innerHTML = resultHTML;

}

function createHistoryCard(algorithm) {

return `

<div class="history-card">

<h4>Запрос #${algorithm.id}</h4>

<p><strong>Текст:</strong> ${algorithm.text}</p>

<p><strong>Подстрока:</strong> ${algorithm.substring}</p>

<p><strong>Результат:</strong> ${algorithm.result.join(', ') || 'не найдено'}</p>

</div>

`;

}

async function loadHistory() {

try {

const token = localStorage.getItem('token');

const [userResponse, allResponse] = await Promise.all([

fetch('http://localhost:8080/api/algorithm', {

headers: { 'Authorization': `Bearer ${token}` }

}),

fetch('http://localhost:8080/api/algorithm/all', {

headers: { 'Authorization': `Bearer ${token}` }

})

]);

if (!userResponse.ok || !allResponse.ok) throw new Error('Ошибка загрузки истории');

const userHistory = await userResponse.json();

const allHistory = await allResponse.json();

userHistoryDiv.innerHTML = userHistory.map(createHistoryCard).join('');

allHistoryDiv.innerHTML = allHistory.map(createHistoryCard).join('');

} catch (error) {

showError(error.message);

}

}

loadHistory();

});

document.addEventListener('DOMContentLoaded', () => {

const tabs = document.querySelectorAll('.tab');

const forms = document.querySelectorAll('.auth-form');

const loginForm = document.getElementById('loginForm');

const registerForm = document.getElementById('registerForm');

tabs.forEach(tab => {

tab.addEventListener('click', () => {

const tabName = tab.dataset.tab;

tabs.forEach(t => t.classList.remove('active'));

tab.classList.add('active');

forms.forEach(form => {

form.classList.remove('active');

if(form.id === `${tabName}Form`) {

form.classList.add('active');

}

});

});

});

loginForm.addEventListener('submit', async (e) => {

e.preventDefault();

const errorElement = document.getElementById('loginError');

errorElement.textContent = '';

const username = document.getElementById('loginUsername').value.trim();

const password = document.getElementById('loginPassword').value.trim();

if(!username || !password) {

errorElement.textContent = 'Заполните все поля';

return;

}

try {

const response = await fetch('/api/auth/login', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ username, password })

});

if(!response.ok) {

const error = await response.json();

throw new Error(error.message || 'Ошибка авторизации');

}

const { accessToken } = await response.json();

localStorage.setItem('token', accessToken);

window.location.href = '/';

} catch (error) {

errorElement.textContent = error.message;

}

});

registerForm.addEventListener('submit', async (e) => {

e.preventDefault();

const errorElement = document.getElementById('registerError');

errorElement.textContent = '';

const username = document.getElementById('registerUsername').value.trim();

const password = document.getElementById('registerPassword').value.trim();

try {

const response = await fetch('/api/auth/register', {

method: 'POST',

headers: { 'Content-Type': 'application/json' },

body: JSON.stringify({ username, password })

});

if(!response.ok) {

const error = await response.json();

throw new Error(error.message || 'Ошибка регистрации');

}

errorElement.textContent = 'Регистрация успешна! Теперь войдите';

setTimeout(() => {

document.querySelector('.tab[data-tab="login"]').click();

}, 2000);

} catch (error) {

errorElement.textContent = error.message;

}

});

if(localStorage.getItem('token')) {

window.location.href = '/';

}

});