Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

Шалавин Константин Сергеевич

Проверка программного кода на плагиат

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Научный руководитель:
к.ф.-м.н., доцент
Т.Ю. Грацианова

Москва, 2023

Аннотация

В работе рассмотрены существующие подходы к обнаружению заимствований программного кода: алгоритм шинглов, подсчет метрик коэффициента Отиаи, расстояния Левенштейна, расстояния Жаккара. Также разработан метод ранжирования программ студентов младших курсов по сходству текста исходного кода.

Оглавление

ВВЕДЕНИЕ

В данной работе речь пойдёт о так называемом **плагиате программного кода.** Начнём со строгого определения *плагиата*. Вот такое определение плагиата дается в толковом словаре: ПЛАГИАТ (от лат. plagio — похищаю) — умышленное присвоение авторства на чужое произведение литературы, науки, искусства, изобретение или рационализаторское предложение (полностью или частично) [1]. Иными словами, плагиат — это выдача другой, уже проделанной работы за свою новую. Плагиаты бывают в разнообразных областях. С развитием Интернета возросла актуальность этой проблемы, ведь при написании каких-либо работ стало проще копировать чужие труды и выдавать их за свои.

Что дает поиск плагиата обществу? Существует мнение, что плагиат является нежелательным, поскольку люди стремятся определить оригинального автора работы. Правила академической честности нужны в научно-исследовательской работе и в работе студента. В случае научно-исследовательской работы, нужно знать первоисточник, чтобы понимать, откуда пришла информация и иметь возможность проверить ее достоверность. Студентов оценивают по их собственным идеям и их самостоятельной работе. Многие учебные курсы и проекты требуют от студентов написания компьютерных программ. Плагиат исходного кода сильно усложняет процесс обучения. Некоторым учащимся проще заниматься плагиатом, чем тратить время на решение задач. Это вынуждает преподавателей еще и анализировать работы на заимствования, а не только сосредоточиться на обучении студентов и исправлении их ошибок. Обнаружение плагиата исходного кода очень важно не только для справедливости между учащимися, но и для того, чтобы они могли достичь результатов обучения, выполняя свои задания. Нахождение плагиата программного кода имеет огромное значение не только с точки зрения справедливости между учащимися, но и для их эффективного обучения путем самостоятельного выполнения заданий. Поэтому поиск плагиата исходного кода становится очень важным.

Из-за большого количества исходных кодов, доступных в Интернете, и исходных кодов, доступных для того же курса в предыдущие годы, задача ручного сравнения каждой пары исходных кодов становится невыполнимой, поскольку количество всех возможных пар N файлов составляет: $N * (N - 1)/2$ (например, если N = 3000, то количество сравниваемых пар программ равно 4,5 миллиона пар). Время вычисления

растет пропорционально квадрату количества файлов. Поэтому возникает потребность в быстрых и точных автоматических инструментах для обнаружения плагиата исходного кода. Но не все программы с высоким сходством являются результатом плагиата. Вот почему преподаватели должны вручную просматривать результаты детекторов плагиата и давать окончательное решение. Таким образом, основная цель инструментов борьбы с плагиатом — сократить количество пар программ (т. е. оставить только очень похожие пары), которые должны быть проверены преподавателями вручную [2]. Проведение таких проверок и аудит их результатов позволяют принять решение о дальнейших действиях, например, дополнительном анализе студенческих работ. Сообщив студентам о результатах проверки, необходимо дать им обратную связь, объяснив, как они могут исправить ошибки и избежать плагиата в будущем.

Существует несколько методов обнаружения плагиата в текстах:

1. Ручной поиск.

Этот метод требует больше времени и усилий, но может быть более точным. Он заключается в том, чтобы вручную сравнивать два или более текста и искать сходства в структуре, фразах, словах и т.д.

2. Использование программ–антиплагиаторов.

“Выявление плагиата в тексте проводится с помощью компьютерных программ, созданных по специальному алгоритму. Они помогают найти заимствования в текстовом документе, оценить качество работы и понять, какие именно фрагменты нуждаются в доработке. Для того, чтобы убедиться, что текст является уникальным, и подобные работы ранее не были выложены в сети Интернет, необходимо скопировать целую статью, вставить ее в специальное окно поиска, нажать кнопку «Проверить», и уже через несколько секунд можно будет оценить процент оригинальности работы. Программа на определение плагиата позволяет избавиться от заимствований в тексте, сделать его качественным и уникальным”, — предлагает услуги один из многих онлайн-сервисов обнаружения плагиата [3]. Существуют различные специализированные программы, которые позволяют проверять текст на уникальность. Выделим три из них:

- antiplagiat.ru [4]
- eTXT [5]
- Text.ru [6]

antiplagiat.ru – единственная система, рекомендованная Министерством образования [7] для использования в учебных заведениях. Данный сервис проверяет текст

на заимствования, делая большой упор на курсовые и дипломные работы, представленные в Интернете. ETXT и Text.ru хороши для использования блогерами и копирайтерами, т.к. любое заимствование из сети может повлечь к различным санкциям от авторов. Отзывы на работу данных сервисов довольно положительные.

Такие системы работают на основе анализа текстов на сходство с другими текстами, хранящимися в специализированных базах данных, которые содержат информацию о публикациях и источниках. Метод использования программ-антиплагиаторов требует доступа к таким базам данных, состоящим из миллионов страниц веб-сайтов, журналов, научных работ, студенческих эссе и других документов.

3. Использование средств машинного обучения. При этом нужно интерпретировать задачу поиска неоднородностей как задачу одноклассовой классификации, а возможное заимствование определять путем обучения и тестирования на фрагментах текста методом кроссвалидации [8]. Метод включает анализ текстов с помощью алгоритмов машинного обучения, которые определяют сходства между текстами, используя базы данных научных статей, специализированные академические базы данных, интернет-корпусы, а также собранные и структурированные наборы данных с веб-сайтов и онлайн-ресурсов. Такой способ детекции может быть более точным, чем ручной поиск, но требует определенных знаний в области машинного обучения.

Плагиат программного кода – частный случай плагиата текста. Детекторы плагиата компьютерных программ работают на основе методов сравнения текстовых файлов. Обычно с программой сопоставляют так называемый исходный код и исполняемый код (а также объектный код, как промежуточный этап). Материалом для проверки на плагиат может являться программа в каком-то из ее представлений. В частности, существенно разные подходы используются при анализе исходного и исполняемого кода программы. Исходный код программы проверять на уникальность легче, поскольку в нем сохраняется больше характеристик, свойственных конкретному автору (в основном это касается стилистических особенностей, которые при компиляции в основном утрачиваются). Тем не менее, и по исполняемому коду тоже можно искать сходства, есть индивидуальная информация, которая хранится и там (используемые алгоритмы, специфические ошибки, способ организации данных).

Методы для обнаружения плагиата в тексте и в программном коде имеют сходства, так как в обоих случаях происходит проверка текста, но могут быть и различия.

Вот несколько сервисов проверки плагиата в программном коде:

- JPlag [9]
- MOSS [10]
- Copyleaks [11]

Данные специализированные системы анализируют и сравнивают программы разных авторов, выявляя совпадения в коде.

Особенность проверки уникальности компьютерных программ студентов младших курсов заключается в том, что студенческие работы содержат разные варианты решения одних и тех же задач. Чтобы найти сходства, необходимо сравнивать программы между собой. Важно учитывать, что студенты могут использовать материалы различных онлайн-ресурсов и библиотек, что также требует особого внимания проверяющих.

В случае программного кода плагиатом может считаться код, по внешнему виду совершенно отличающийся от авторского (например, написанный на другом языке), но действующий точно по такому же алгоритму, работающий точно таким же образом. В случае с текстами тоже говорят о том, что можно «украсть идею», но все-таки реже. В студенческих программах мы о таком плагиате говорить не будем, так как на младших курсах речь о воровстве алгоритмов не идет, потому что идея решения чаще всего одна и та же. К тому же все программы пишутся в основном на одном языке.

1 Постановка задачи

Цель данной работы – изучение и разработка методов и средств проверки уникальности компьютерных программ и применение этих методов для анализа сходства программного кода студентов младших курсов. Ввиду этого объектом исследования являются маленькие компьютерные программы.

Можно выделить следующие подзадачи:

- Выполнить обзор существующих автоматических детекторов плагиата в программах и выявить возможности их применения для поставленной задачи. Исследовать основные типы заимствований исходного кода и методы их обнаружения.
- Разработать алгоритм поиска сходства программного кода и создать тестовый набор программ для первичного анализа этого алгоритма и его оптимизации.

2 Обзор существующих решений

Системы по обнаружению плагиата программного кода являются частным случаем задачи обнаружения плагиата текста. Сходства и различия алгоритмов для детекции определяются целями и особенностями обработки текстовых файлов и компьютерных программ. Для плагиата текста чаще всего текст заимствуется только кусками. А в случае с программами код копируется целиком, чтобы сохранить работоспособность. Далее программа преобразуется целиком или некоторым образом частично для того, чтобы выдать решение за оригинальное.

Сходства систем по обнаружению плагиата текста и программного кода:

- Использование методов сравнения двух текстов или кодовых файлов.
- Необходимость определить, насколько сильно один текст или код похож на другой.

Отличия алгоритмов при детекции плагиата текста от алгоритмов детекции плагиата программного кода:

- Алгоритмы для обнаружения плагиата в тексте и в программном коде используют разные методы сравнения. В случае с текстом часто используются векторный анализ, где каждый текст представляется в виде вектора. Для сравнения программного кода используются алгоритмы, основанные на синтаксическом анализе, сравнении токенов, анализе структуры программы.
- Типы ошибок: алгоритмы антиплагиата для текста могут допускать ошибки при проверке синонимов, сленга, цитат и т.д. Детекторы программного кода могут допускать ошибки при проверке наличия аналогичных алгоритмов и методов. Некоторые способы плагиата, такие как переименование переменных или функций, могут затруднить выявление заимствования, поскольку это может изменить внешний вид кода, но при этом сохранить его структуру и алгоритм.
- Уровень сложности: алгоритмы антиплагиата для программного кода более сложные, чем для текста, так как программный код более структурирован и имеет элементы, которые могут быть использованы в разных программах.
- Алгоритмы для обнаружения плагиата в тексте не всегда могут корректно обработать перефразирования, что может привести к неверному результату. В то же время, детекторы для программного кода не всегда могут корректно

обработать изменения в исходном коде, которые могут быть вызваны техническими причинами. Такие изменения в коде могут включать добавление новых функций, удаление устаревших кодовых блоков, изменение переменных или переименование функций. Если детектор плагиата не учитывает такие изменения или неправильно интерпретирует их, он может дать ложные положительные результаты, указывая на наличие плагиата там, где его фактически нет.

В обзоре представлены готовые сервисы для проверки уникальности компьютерных программ. Исследованы этапы обнаружения заимствований в программном коде: выбор метода, предобработка программы перед проверкой, определение сходства между программами.

2.1 Общедоступные системы проверки кода на плагиат

Для решения задачи выявления заимствований в коде существуют определенные подходы. Почти все они работают либо с лимитированным числом языков, либо сканируют интернет на похожие куски кода.

Существует несколько доступных систем обнаружения плагиата в программном коде, которые могут использоваться для проверки уникальности кода. Эти системы обладают своими особенностями и могут использоваться для различных целей. Некоторые из них являются коммерческими, в то время как другие доступны бесплатно, и выбор конкретной системы будет зависеть от потребностей пользователя. На текущий момент наиболее известны три основные системы для обнаружения заимствования программного кода. Рассмотрим их детально.

2.1.1 Система JPlag

JPlag [9] — это программное обеспечение для обнаружения плагиата, которое используется в основном для проверки исходного кода программ на наличие сходств и заимствований из других источников. JPlag разработан Гвидо Мальполем в 1996 году в Техническом институте Карлсруэ [12]. Изначально JPlag предназначался для нахождения плагиата среди упражнений студентов, на них же он и тестировался. Детектор поддерживает языки: Java, C, C++, Scheme. Алгоритм. Алгоритм работает в 2 этапа:

1. Все программы, которые надо сравнить, синтаксически анализируются и преобразуются в последовательность токенов.
2. Эти токены попарно сравниваются, для каждой пары определяется "похожесть". Это делается алгоритмом *жадного строкового замощения* [13].

В завершении JPlag выводит в виде HTML страниц найденные сходные подстроки для возможности дальнейшего анализа. В техническом отчете утверждается, что по результатам тестирования 12 различных наборов Java программ, JPlag находит весь плагиат, за небольшими исключениями. При этом время выполнения меньше одной минуты на запрос из 100 программ, в каждой из которых по несколько сотен строк.

JPlag также имеет некоторые дополнительные функции, такие как поддержка режима командной строки для автоматической проверки большого количества файлов, а также возможность генерировать графические отчеты в виде диаграмм и графиков, что позволяет быстро оценить общее количество заимствований в проверяемом коде.

2.1.2 Система MOSS

MOSS (Measure of Software Similarity) [10] — это автоматизированная система обнаружения плагиата, которая был разработана в 1994 году Алексом Эйкеном в Университете Стэнфорда. Поддерживает языки: C, C++, Java, C#, Python, Visual Basic, Javascript, FORTRAN, ML, Haskell, Lisp, Scheme, Pascal, Modula2, Ada, Perl, TCL, Matlab, VHDL, Verilog, Spice, MIPS assembly, a8086 assembly, a8086 assembly, MIPS assembly, HCL2.

Алгоритм MOSS использует *метод просеивания* для построения идентификационных меток (отпечатков) [14]. Он работает на основе сравнения кода на уровне строк и токенов, что позволяет обнаружить даже измененные или переименованные переменные и функции.

Утверждается, что алгоритм, на котором основан MOSS, модифицирован против всех известных алгоритмов обмана детекторов [12].

Принцип работы MOSS:

1. Пользователь загружает программный код. MOSS анализирует код, разбивая его на блоки и строит для каждого блока хеш-код.
2. Детектор сравнивает каждый блок кода, используя алгоритмы, которые определяют схожесть блоков, даже если они несколько отличаются друг от друга.

MOSS также учитывает комментарии, форматирование и прочие незначительные различия.

3. MOSS собирает все блоки кода, которые он нашел похожими, в группы. Каждая группа содержит блоки кода, которые находятся в различных программах, но имеют схожую структуру или используют похожие алгоритмы.
4. Система возвращает список файлов, которые могут содержать плагиат и процентное соотношение между сравниваемыми файлами. MOSS также позволяет просмотреть подробную информацию о том, какие строки кода являются похожими, и общее количество совпадений в каждом файле.

Детектор MOSS может обрабатывать большие наборы данных и работать с большим количеством файлов программного кода, имеет настраиваемые параметры сравнения, которые позволяют отрегулировать систему для различных типов проверяемых программ.

2.1.3 Система Copyleaks

Copyleaks [11] — это облачный онлайн-сервис для обнаружения плагиата в тексте, включая программный код. Система использует AI-технологии для обнаружения плагиата в различных типах контента, включая текст, документы, веб-страницы, а также программный код на разных языках программирования Алгоритмы этого сервиса скрыты от пользователей и активно развиваются.

Одним из ключевых преимуществ Copyleaks является его способность поддерживать множество языков программирования, включая Java, C++, C#, Python, Ruby и другие. Система использует алгоритмы машинного обучения для анализа текстов и выявления сходств и расхождений между ними. Она также предлагает инструменты для анализа плагиата в режиме реального времени и поиска сходств между введенным текстом и веб-страницами.

Copyleaks имеет как бесплатную, так и платную версии с различными функциями и возможностями. Кроме того, система предоставляет API для интеграции с другими приложениями и системами.

Механизм работы Copyleaks для обнаружения плагиата в программном коде следующий:

1. Пользователь загружает программный код, который нужно проверить на наличие плагиата, на сервер Copyleaks.

2. Copyleaks использует алгоритмы машинного обучения, чтобы улучшить точность обнаружения плагиата в программном коде. Он определяет сходство не только по ключевым словам и фразам, но и по структуре программного кода, что позволяет выявлять даже неочевидные формы плагиата.
3. Пользователи могут просматривать отчеты о проверке плагиата в интерфейсе Copyleaks и получать информацию о найденных сходствах в своих программах.

Таким образом, Copyleaks предоставляет надежный и эффективный способ обнаружения плагиата в программном коде с помощью современных технологий машинного обучения и искусственного интеллекта.

2.1.4 Сравнение общедоступных систем

Автоматические детекторы плагиата, такие как JPlag, MOSS plagiarism detection и CopyLeaks, могут быть полезными для обнаружения совпадений в небольших студенческих программах. Все три детектора сравнивают файлы, присылаемые пользователем, между собой. Одним из главных преимуществ является их способность быстро и точно анализировать большое количество кода, что может сэкономить много времени и усилий проверяющих.

Недостатком этих систем является то, что они могут давать ложно-положительные результаты или пропускать настоящий плагиат, что может быть вызвано общими алгоритмами обработки текста. Это может привести к тому, что преподаватели и ассистенты тратят дополнительное время, проверяя, является ли фрагмент кода действительно плагиатом.

JPlag является бесплатной системой обнаружения плагиата в коде, что является ее большим преимуществом. Недостатком JPlag является ограниченный набор поддерживаемых языков программирования, что может быть проблемой для студентов, использующих редкие языки программирования. Ограничением является отсутствие графического интерфейса, так как его использование требует некоторых навыков работы в командной строке. Однако JPlag имеет открытый исходный код, что позволяет пользователям настраивать систему для своих потребностей.

MOSS при использовании для академических и научных целей не требует оплаты. Система имеет графический интерфейс, что делает ее более доступной для пользователей. Система может работать с большим объемом данных и обеспечивать высокую точность обнаружения плагиата в больших наборах кода. Так же, как и JPlag,

она является общедоступной системой с открытым исходным кодом, что позволяет исследователям и разработчикам программного обеспечения улучшать функциональность и расширять ее возможности.

Copyleaks является коммерческой системой обнаружения плагиата в коде. Однако есть бесплатная пробная версия, которая позволяет пользователю ознакомиться с основными функциями системы. Copyleaks также имеет графический интерфейс. Этот сервис предоставляет API и SDK для интеграции с различными инструментами и платформами разработки программного обеспечения. Это позволяет разработчикам внедрить функциональность обнаружения плагиата Copyleaks в свои инструменты и процессы разработки программного обеспечения. Кроме того, Copyleaks также может работать с платформами управления контентом и облачными хранилищами данных, такими как Dropbox, Google Drive и OneDrive.

В Таблице 2.1 представлено сравнение данных детекторов плагиата.

Детектор	Коммерческая составляющая	Графический интерфейс	Число языков программирования	Ограничения на объем кода	Точность на малых объемах кода	Точность на больших объемах кода	Открытый исходный код	Интеграции с инструментами и платформами	AI
JPlag	–	–	12	+	+	+	+	–	–
MOSS	–	+	более 60	–	+	+ (наибольшая)	+	–	–
CopyLeaks	+	+	более 100	–	зависит от настроек системы	+	–	+	+

(Таблица 2.1) Сравнение возможностей использования, преимуществ и недостатков автоматических детекторов плагиата.

Эффективность детекторов плагиата (способность обнаруживать неоригинальный код и точность определения доли скопированного кода относительно всего объема проверяемого кода) может сильно варьироваться в зависимости от различных факторов, таких как объем кода, сложность алгоритмов, типы сравнений, используемые языки программирования, качество анализа и многие другие.

В целом, детекторы плагиата являются полезным инструментом для выявления неоригинального кода, особенно в больших проектах и при проверке работы студентов.

Но они не могут гарантировать 100% точность и надежность в выявлении плагиата, так как существуют методы обхода детекторов, а также возможны ложные срабатывания.

Поэтому при использовании детекторов плагиата важно учитывать их ограничения и сочетать их с другими методами проверки оригинальности кода, такими как ручная проверка, использование других детекторов и т.д. Создание в данной работе новой системы обнаружения сходства обосновано тем соображением, что в новом детекторе можно улучшить производительность, точность и удобство использования именно при работе с маленькими студенческими программами.

2.2 Типы плагиата программного кода и методы их обнаружения

Условное разделение на группы для плагиата программного кода представлено в статье [15]:

1. Простое копирование программного кода без каких-либо изменений, то есть он идентичен оригинальному коду, включая комментарии.
2. Копирование кода с незначительными изменениями, например, изменение идентификаторов функций и переменных, типов данных или строковых литералов.
3. Копирование кода с модификациями, такими как добавление, редактирование или удаление фрагментов кода или изменение порядка их исполнения, но без влияния на логику программы.
4. Переписывание программы с сохранением логики и функциональности, но с абсолютно отличной синтаксической структурой от оригинала.

Автор статьи [15] утверждает, что качественная система антиплагиата должна хорошо определять первые три типа плагиата, тогда как выявление заимствований четвертого типа крайне затруднительно и часто не является плагиатом, так как может быть результатом копирования алгоритма, а не конкретных фрагментов кода. Также приводятся 5 наиболее универсальных и распространенных подходов (по мнению автора):

- Text-based метод заключается в сравнении текстовых представлений программ на основе некоторой метрики, например, расстояния Левенштейна или Джаро-Виклера. Данный способ отличается своей скоростью и простотой, но

результативность быстро снижается даже на простейших "косметических" модификациях.

- Token-based метод основан на преобразовании ключевых слов программы в последовательность лексем языка программирования (далее просто токенов). Полученные токены сравниваются любым доступным способом. Этот алгоритм гораздо точнее предыдущего, так как игнорирует все изменения второго типа, тем не менее он всё ещё не справляется со структурными изменениями.
- Metric-based метод определяет на полученных в предыдущем методе токенах некоторые метрики (например, кол-во используемых циклов и условных конструкций), а далее считает схожесть программ на основе количества совпадающих метрик. В целом алгоритм отлично дополняет другие методы антиплагиата, однако он часто даёт ошибочный результат (в частности, метод успешно отрабатывает на программах с переименованием переменных и изменением условий и циклов, но моментально деградирует при добавлении в код NO-OPов по типу объявления пустого цикла или инициализации неиспользуемых значений).
- Tree-based подход требует представления кода в виде абстрактного синтаксического дерева (далее просто AST, Abstract Syntax Tree). В таком формате программы сравниваются любым доступным способом. Например "наивным" подсчетом совпадающих узлов. Часто данный способ считается наиболее эффективным, но в то же время его сложнее внедрить: помимо построения самих AST, нужна реализация алгоритма для их сравнения.
- Существуют системы, которые проверяют работу алгоритма исходного кода. Такой метод называется Binary-based. А алгоритмы решения одной и той же задачи у разных студентов всегда очень близки, поэтому такой метод не подходит для проверки студенческих работ на плагиат.

Для исследования эффективности алгоритмов поиска сходства была использована небольшая синтетическая выборка, включающая некоторые модификации программ на языках Java и C++ (~750 символов) для всех четырех типов заимствований. И вот как показали себя примененные методы по отдельности (коэф. схожести усреднен и округлен):

Модификация / процент схожести программ	Text-based	Token-based	Metric-based	Tree-based	Binary-based
Изменение комментариев (тип 1)	100%	100%	100%	100%	99%
Реформатирование кода (тип 2)	97%	97%	92%	93%	100%
Добавление неиспользуемых зависимостей (тип 2)	89%	92%	85%	91%	99%
Переименование идентификаторов (тип 2)	85%	100%	100%	96%	98%
Изменение типов данных (тип 2-3)	97%	99%	100%	98%	85%
Изменение порядка исполнения кода (тип 3)	78%	86%	95%	91%	81%
Выделение частей кода в функции (тип 3-4)	65%	74%	72%	51%	67%

(Таблица 2.2) Эффективность методов обнаружения плагиата в зависимости от типа плагиата [15].

В Таблице 2.2. важна динамика поведения различных методов. Лучше всего себя проявляют Token-based и Metric-based методы — их коэффициенты схожести не так быстро уменьшаются с усложнением модификации проверяемых исходных программ.

Есть множественные модификации каждого из методов, так или иначе повышающие эффективность скоринга. В этом плане интересен *метод шинглов*, а также совершенно другие подходы, которые могут основываться, например, на поведении программы во время исполнения (behavior-based метод) ини на графе её зависимостей (PDG-based метод) [15].

Подробнее рассмотрим *метод шинглов*. Реализация алгоритма подразумевает несколько этапов:

1. Канонизация текстов. Из модели, с большим количеством избыточной информации, переходят в более компактную модель, где незначимая информация удалена.
2. Разбиение текста на шинглы. Необходимо разбить каждый из канонизированных текстов на подпоследовательности — шинглы (N-граммы).
3. Нахождение набора контрольных сумм шинглов.

4. Вычисление различных мер сходства (метрик) между двумя наборами контрольных сумм.

Плюсы: различные языки не требуют изменений в исходном коде, либо изменения минимальны, поэтому возможно добавление новых языков программирования, т.к. алгоритм универсален для всех языков.

Проблема алгоритма заключается в количестве сравнений, ведь это напрямую отражается на производительности. Увеличение количества шинглов для сравнения характеризуется экспоненциальным ростом операций, что критически отразится на производительности.

От выбора длины шингла также напрямую зависит результат сравнения. Поэтому необходимо определить длину шингла, при которой заимствования будут заметны, но при этом для разного кода мы не наблюдали бы совпадений.

Для реализации задачи, поставленной в данной работе, среди всех методов детекции был выбран *метод шинглов*, как предположительно наиболее эффективный.

2.3 Обзор некоторых метрик

Расстояние Левенштейна — метрика, измеряющая по модулю разность между двумя последовательностями слов. Она определяется как минимальное количество однословных операций (а именно вставки, удаления, замены), необходимых для превращения одной последовательности слов в другую [16].

Пусть S_1 и S_2 — две строки (длиной M и N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) d(S_1, S_2) можно подсчитать по следующей рекуррентной формуле d(S_1, S_2)=D(M,N)

$$D(i,j) = \begin{cases} 0, & i=0,\ j=0 \\ i, & j=0,\ i>0 \\ j, & i=0,\ j>0 \\ \min\{ & \\ \quad D(i,j-1)+1, & \\ \quad D(i-1,j)+1, & j>0,\ i>0 \\ \quad D(i-1,j-1)+\mathrm{m}(S_1[i],S_2[j]) & \\ \} & \end{cases},$$

где m(a,b) равна нулю, если a=b и единице в противном случае; min{a,b,c}

возвращает наименьший из аргументов. Здесь шаг по i символизирует удаление (D) из первой строки, по j — вставку (I) в первую строку, а шаг по обоим индексам символизирует замену символа (R) или отсутствие изменений (M).

Расстояние Жаккара. Коэффициент (индекс) сходства Жаккара сравнивает элементы из двух текстов и определяет, какие из них являются общими, а какие – различными. Для улучшения метода обычно текст исходного кода программ представляют в виде n-грамм. Коэффициент Жаккара равен количеству общих элементов, деленному на количество всех элементов в обоих текстах, или, иначе говоря, пересечению двух текстов, деленному на их объединение. Мера показывает, насколько похожи два текста, и принимает значения от 0 до 1. Расстояние Жаккара показывает, насколько различны два текста. Оно является дополнением к коэффициенту Жаккара и может быть найдено вычитанием коэффициента Жаккара из 1 [17].

$$K = \frac{|A \cap B|}{|A \cup B|}$$

Коэффициент Отиаи. Коэффициент Отиаи (мера Отиаи, коэффициент Оцуки-Отиаи, коэффициент Отиаи-Баркмана, косинусный коэффициент, геометрический коэффициент) — бинарная мера сходства, предложенная японским биологом Акирой Отиаи в 1957 году, в дальнейшем обобщенная и нашедшая применение в разнообразных приложениях и за рамками биологии. В стандартном определении коэффициент для двух произвольных множеств A и B вводится следующим образом:

$$K = \frac{|A \cap B|}{\sqrt{|A| \cdot |B|}},$$

где $|A|$ — мощность множества A [18].

Исследование эффективности алгоритмов поиска сходства на основе подсчета отдельных метрик было проведено в работе “Методы поиска плагиата в кодах программ” [17].

Результаты анализа показаны в Таблице 2.3

Метрика	**Время**	**Память**	**Вероятность найти похожие программы**	**Вероятность, что похожие программы действительно похожи**
Численные значения атрибутов	Квадратичное	Линейная	Высокая	Низкая
Наибольшая общая подпоследовательность	Квадратичное	Линейная	Очень высокая	Низкая
Расстояние Жаккара	Квадратичное	Линейная	Высокая	Высокая
Расстояние Левенштейна	Квадратичное	Квадратичная, сводится к O(min{m,n})	Очень высокая	Высокая
Расстояние Колмогорова	Сверхлинейное	—	—	—

(Таблица 2.3) Результаты исследования алгоритмов для обнаружения плагиата, использующих отдельные метрики [17].

Наилучшее время показывает расстояние Колмогорова, однако его невозможно реализовать на практике. Затраты памяти наихудшие у расстояния Левенштейна. Следует отметить, что для выполнения задач важно качество преобразований текста, а затраты менее важны. Наивысшую вероятность найти похожие программы показывает расстояние Левенштейна.

Какими должны быть метрики? В статье [19] Вейл сформулировал какими качествами должны обладать метрики, чтобы быть полезными на практике:

- Они должны представлять такие характеристики, которые достаточно трудно изменить, пытаясь замаскировать копию.
- Должны быть устойчивы к незначительным изменениям исходного кода.
- Должно быть просто сравнивать, используя эти метрики.
- Должны быть достаточно общими (чтобы быть применимыми к довольно широкому ряду языков программирования).

Метрики расстояния Левенштейна, расстояния Жаккара и коэффициента Отиаи удовлетворяют всем этим требованиям. Поэтому они были выбраны для разработки алгоритма обнаружения текстового сходства.

3 Описание реализации задачи

В данной работе для оценки схожести использован метод *шинглов*. В рамках поставленной задачи такой метод состоит из следующих этапов:

1. Предобработка программ:
    a. Канонизация.
    b. Разбиение на шинглы.
    c. Вычисление контрольных сумм шинглов. Хеш-функция crc32 из библиотеки binascii [20] языка Python [21] берет шингл и преобразует его в число, которое называется хеш-значением или просто хешем.
2. Расчет коэффициентов сходства: *расстояние Левенштейна*, *расстояние Жаккара* и *коэффициент Отиаи* для пар программ. Эти метрики отображаются в виде процентов.

После обнаружения совпадений методом шинглов программа отбирает пары, у которых коэффициент сходства выше заданного значения и сортирует отобранные пары по убыванию их коэффициентов. Такая система является системой ранжирования компьютерных программ по сходству.

Для проверки работоспособности алгоритма и его отладки разработан тренировочный датасет, содержащий шесть программ, которые для наглядности реализуют решение задачи быстрой сортировки:

1. Программа, содержащая оригинальное решение.
2. Плагиат оригинального решения путем добавления пробельных символов в различных местах.
3. Плагиат оригинального решения путем добавления комментариев в различных местах.
4. Плагиат оригинального решения путем изменения стиля (например: a<b -> b>a; a,b=b,a -> tmp=a,a=b,b=tmp; while -> for и т.д.).
5. Плагиат оригинального решения путем изменения идентификаторов (названий переменных, функций и т.д.).
6. Альтернативное решение данной задачи другим студентом.

Результаты анализа совпадений (заимствований) на тестовом датасете использовались для улучшения предобработки программ и расчета коэффициентов сходства.

3.1 Канонизация

Прежде всего нужно некоторым образом предобработать программы, то есть предварительно перевести их в к единому формату для снижения чувствительности к плагиату второго типа. Отбросив лишнюю информацию и выделив нужные зависимости, переводим программу в более компактное представление. С помощью регулярных выражений производится предобработка текста программ:

- удаление комментариев (с учетом языка программирования);
- очистка кода от пробелов, табуляций и сносов строк;
- получение одинакового разбиения кода на токены в зависимости от стиля написания программы.

В Таблице 3.1 приведены примеры токенов [17].

Имя токена	Описание	Примеры
Идентификаторы	Имена переменных или функций, которые задаёт программист	*x,* tokenizer, digit
Ключевые слова	Имена идентификаторов, которые имеют специальное значение для компиляторов на языке программирования (не могут совпадать с именами переменных)	function, if, for
Разделители	Знаки пунктуации и парные разделители	{, (, ;
Операторы	Набор команд	+, =, <
Литералы	Фиксированные значения некоторого типа данных	True, 1, String

(Таблица 3.1.) Примеры токенов, на которые разбивается текст [17].

Пример разбиения на токены:

a<b -> [“a<b”] -> [“a”, “<”, “b”]

a < b -> [“a”, “<”, “b”] -> [“a”, “<”, “b”]

Далее каждый символ приводится к нижнему регистру для того, чтобы весь текст был приведен к единому формату.

Пример обновления текста с помощью регулярного выражения для очистки от комментариев на языках C и C++:

```
if file_extension in [".c", ".C", ".cpp", ".CPP"]:
    text = re.sub('//.*?\n|/\*.*?\*/', '', text)
```

Для каждого нового языка необходимо добавить регулярное выражение, которое выполнит очистку от комментариев. Это нужно, чтобы улучшить предобработку кода. Написание такого регулярного выражения даёт стопроцентное сходство оригинального решения и решения с добавлением комментариев.

Регулярное выражение для очистки от лишних пробельных символов и для правильного разбиения на токены имеет вид:

```
return re.findall(r"[\w']+", text.replace('\n', ' ').lower())
```

Оригинальное решение и решение с добавлением различных пробельных символов полностью совпадают при использовании этого регулярного выражения.

Метод шинглов сравнивает текстовые строки. Python проводит сравнение чисел быстрее, чем строки, так как числа занимают меньше памяти. Поэтому строковые значения преобразуются в числовые данные с помощью вычисления контрольных сумм (хеширования). Можно вычислять хеш-функции для каждого слова или же для каждой N-граммы кода. Для каждого слова вычисление хеша является частным случаем работы с N-граммами при $N = 1$. В зависимости от N будет разниться и показатель сходства.

Изначально при сравнении методом шинглов при $N = 4$ оригинальное решение и плагиат оригинального решения путем добавления пробельных символов в различных местах имеют коэффициент сходства 56%. После написания регулярного выражения, не просто разбивающего по пробелам, а разделяющего на токены, сходство стало 100%.

3.2 Выбор N-граммы

N-граммы составляются внахлест: первое — N-ное слово, второе — (N+1)-ое слово, третье — (N+2)-ое слово и т.д. Так для участка кода *“for item in range [1, 2, 3, 4]”* канонизированный вид выглядит:

[“for”, “item”, “in”, “range”, “[”, “1”, “,”, “2”, “,”, “3”, “,”, “4”, “]”, “:”]

для N=4 шинглы выглядят:

1: [“for”, “item”, “in”, “range”]

2: [“item”, “in”, “range”, “[”]

3: [“in”, “range”, “[”, “1”]

…

10: [“3”, “,”, “4”, “]”]

11: [“,”, “4”, “]”, “:”]

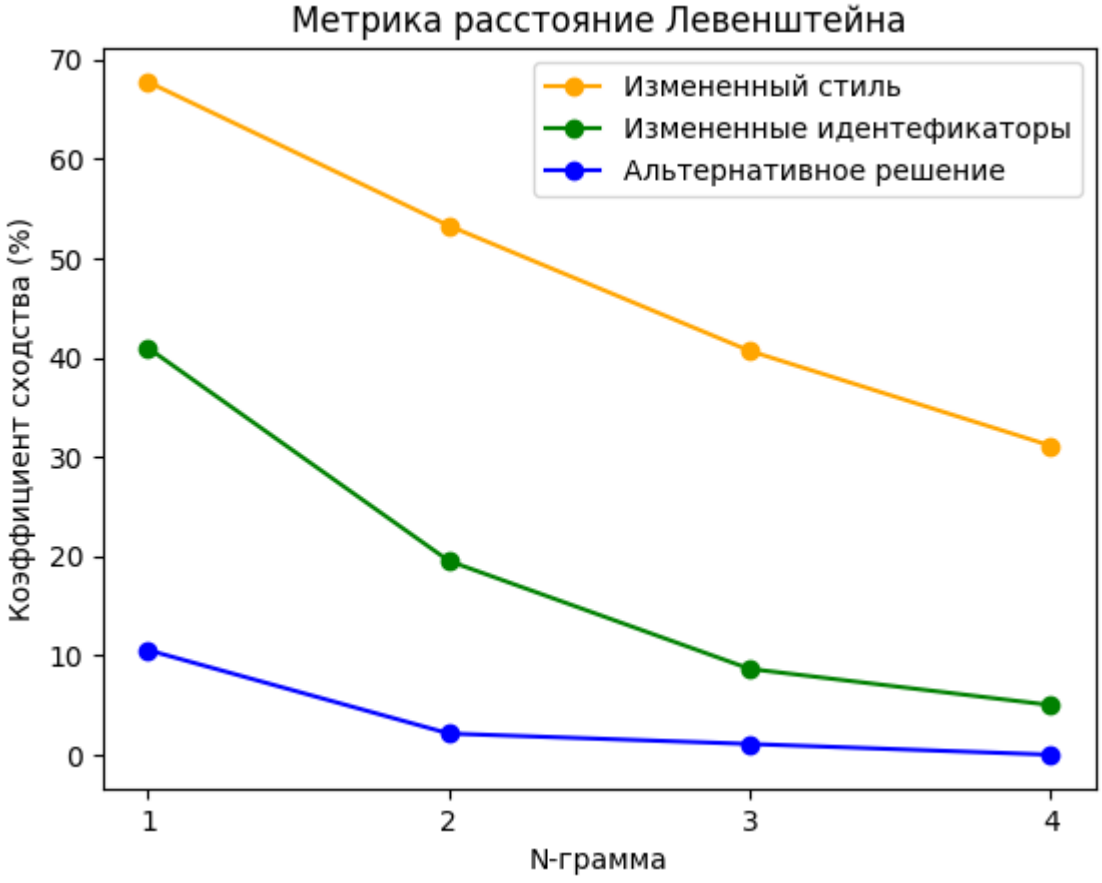
Следующим этапом является выбор размера N-граммы для разбиения канонизированных программ на шинглы.

Ниже приведены результаты сравнения оригинального решения с тремя программами:

- плагиат оригинального решения путем изменения стиля;
- плагиат оригинального решения путем изменения идентификаторов;
- альтернативное решение данной задачи другим студентом.

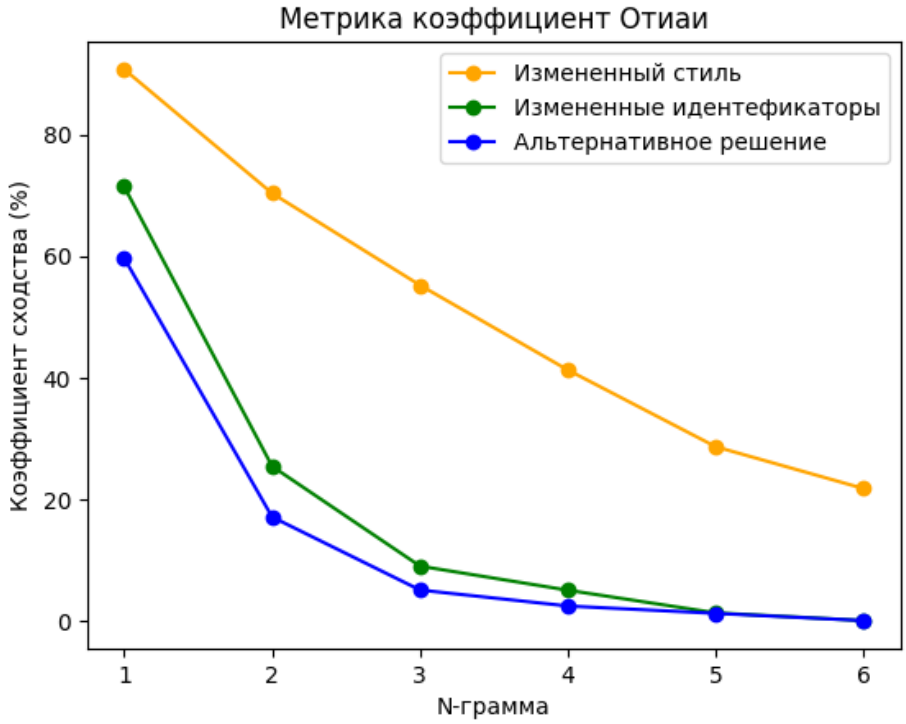
Сравнение выполнено с помощью подсчета методом шинглов значений метрик: *расстояние Левенштейна, расстояние Жаккара* и *коэффициент Отиаи*.

Результаты подсчета метрикой *расстояние Левенштейна* отображены на Рисунке 3.2 для $N = \{1, 2, 3, 4\}$



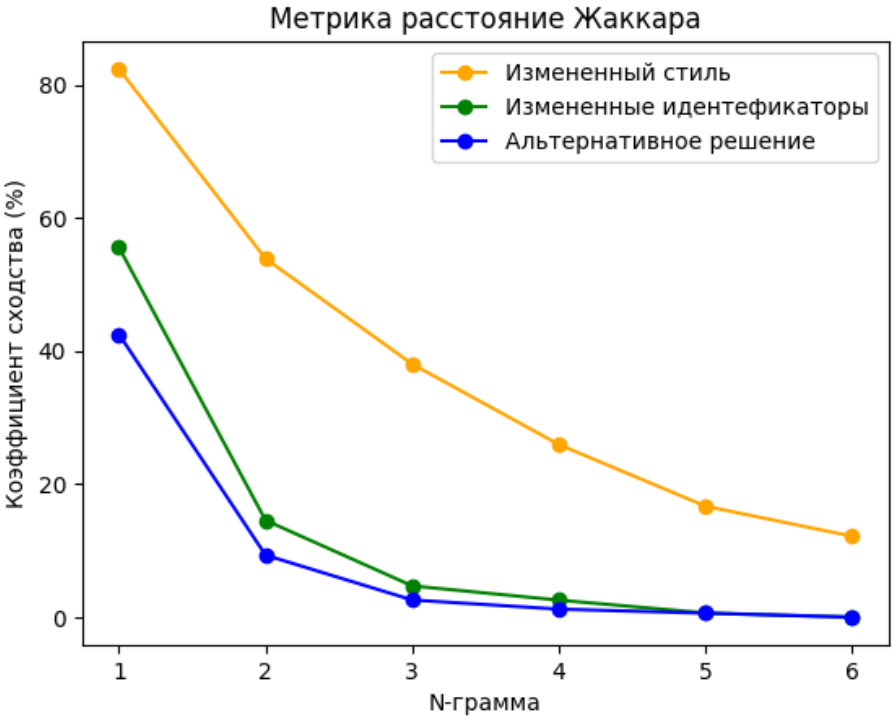
(Рисунок 3.2) Результаты подсчета метрикой расстояние Левенштейна.

Результаты подсчета метрикой *Отиаи* отображены на Рисунке 3.3 для $N = \{1, 2, 3, 4, 5, 6\}$.



(Рисунок 3.3) Результаты подсчета метрикой Отиаи.

Результаты сравнения метрикой *расстояние Жаккара* отображены на Рисунке 3.4 для $N = \{1, 2, 3, 4, 5, 6\}$.



(Рисунок 3.4) Результаты подсчета метрикой расстояние Жаккара.

Анализ полученных данных показывает, что при сравнении оригинальной программы с альтернативным решением *методом шинглов* в случае N=4 значения метрик *расстояние Жаккара*, *коэффициент Отиаи* имеют низкий коэффициент сходства (менее 3%). При этом сходство оригинала с плагиатом путем изменением стиля показало не такой низкий результат, как при N=5 и N=6. Для этих же алгоритмов сходство с плагиатом оригинального решения путем изменения стиля не особенно сильно отличается от сходства оригинала с альтернативным решением.

Подсчет метрикой *расстояние Левенштейна* показывает хорошие результаты для N=3. Показатель сходства оригинального решения с плагиатом путем изменения идентификаторов намного выше сходства оригинала с альтернативным решением, по сравнению с указанными выше метриками. Подсчет метрикой *расстояние Левенштейна* имеет бóльшие затраты по времени и памяти, так как алгоритм более сложный, но имеет более высокий коэффициент сходства.

3.3 Тестирование

При тестировании алгоритма был использован датасет для отладки системы антиплагиата UnPlag [22]. Он состоит из 16 программ на языке C++, реализующих одну задачу по программированию. В 11 из них содержатся уникальные решения. Остальные пять программ являются различными вариациями плагиата одного из вышеуказанных уникальных решений.

На данном тестовом наборе программ алгоритм поиска сходства показал следующие результаты:

- При сравнении одиннадцати уникальных программ между собой не было обнаружено сходство выше 25%. Исключение составила пара программ, реализующих поиск в глубину и поиск в ширину. Их сходство составило 52.68%.
- Сходство программы-первоисточника с программой, плагиат которой выполнен путем изменения порядка блоков, объявлений и т.д., составило 66.49%.
- Сходство программы-первоисточника с программой, плагиат которой выполнен путем добавления бесполезного кода между исходным кодом, составило 81.65%.

- Сходство программы-первоисточника с программой, плагиат которой выполнен путем изменения названий идентификаторов, составило 1%.
- Сходство программы-первоисточника с программой, плагиат которой выполнен путем умеренного плагиата (другой main, замены while на for и т. д.), составило 42.6%.
- Сходство программы-первоисточника с программой, плагиат которой выполнен путем сильного плагиата составило 27.39%.

Анализ ошибок.

1. Плагиат обнаружен во всех случаях, где он и был, кроме варианта, когда плагиат выполнен путем изменения названия идентификаторов. Подход обнаружения на основе текста не зависит от языка программирования, но его можно сбить с толку тривиальной атакой, такой как переименование идентификаторов. Программа-первоисточник имеет низкое сходство с плагиатом такого качества. Такое решение является ложноотрицательным. Это ожидаемый результат для метода шинглов, потому что в случае студенческих заданий код разных студентов чаще всего отличается названиями идентификаторов.
2. Задачи, реализующие поиск в глубину и поиск в ширину, очень похожи, и код очень похож, потому что это решение одного автора. Такое решение является ложноположительным. Это связано с тем, что программисты часто используют свои предыдущие фрагменты кода в новых программах.

Для используемого датасета точность (precision) составляет 80%, как и полнота (recall) 80%.

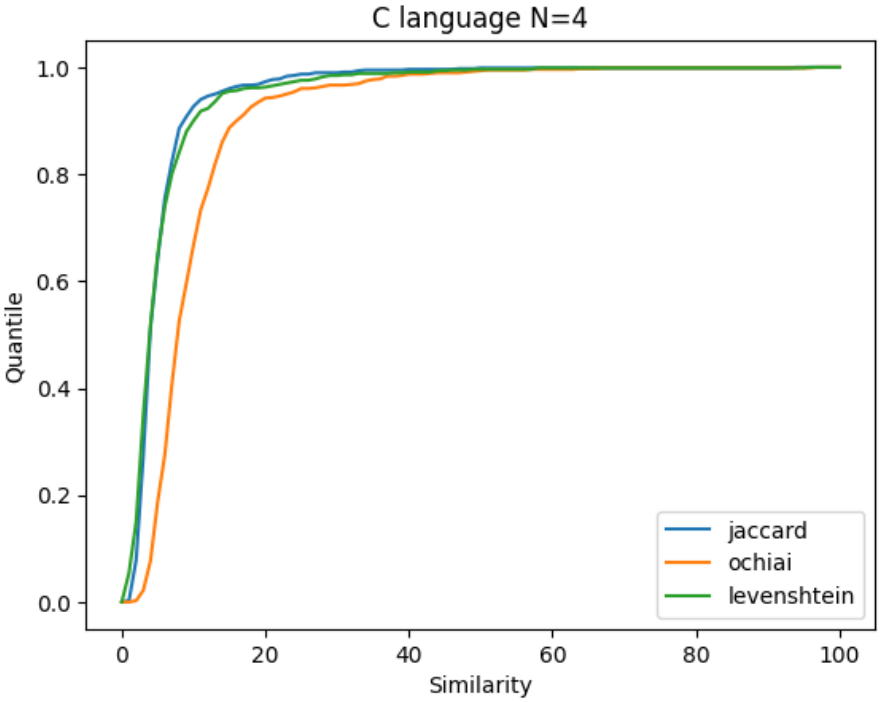
Полученные при тестировании данные подтверждают работоспособность разработанной системы и ее эффективность (точность и полноту).

3.4 Результаты проверки студенческих работ

Для анализа полученного алгоритма был собран архив студенческих программ. Данный набор программ был разделен на группы, в каждой из которых были программы реализующие одну и ту же задачу, написанные на одном и том же языке: Архив 1 (язык C), Архив 2 (язык C) и Архив 3 (язык C++).

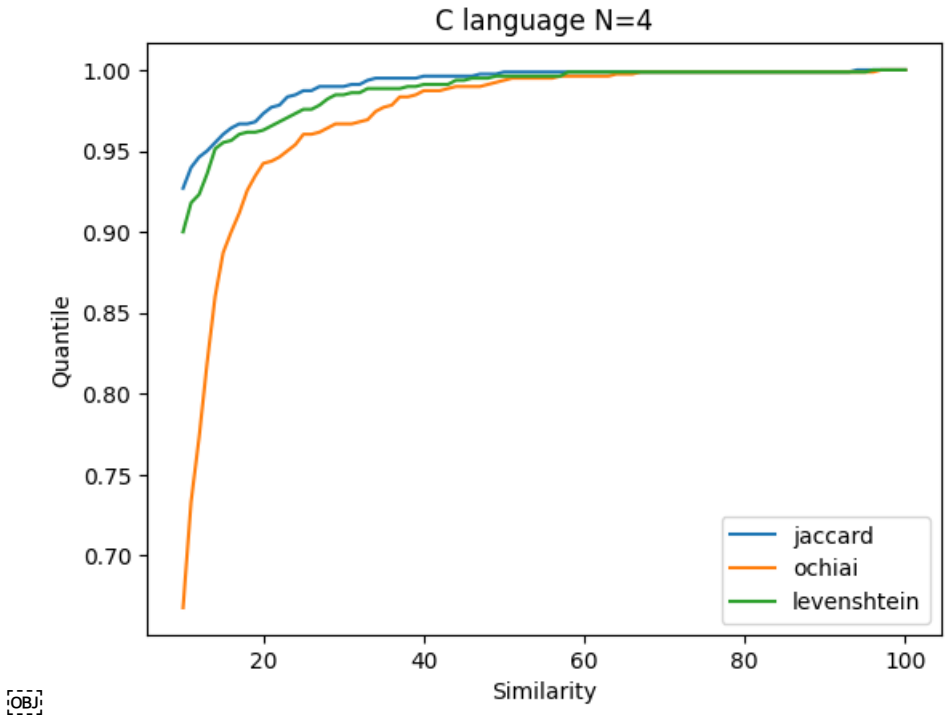
Рассмотрим диаграмму на Рисунке 3.6. Она отображает квантили распределения сходства студенческих программ на языке C из Архива 1. По горизонтальной оси

меняется коэффициент сходства в процентах. По вертикальной оси отображается соответствующий ему квантиль. Диаграмма отображает три ряда данных: для расстояния Жаккара, коэффициента Отиаи и расстояния Левенштейна.

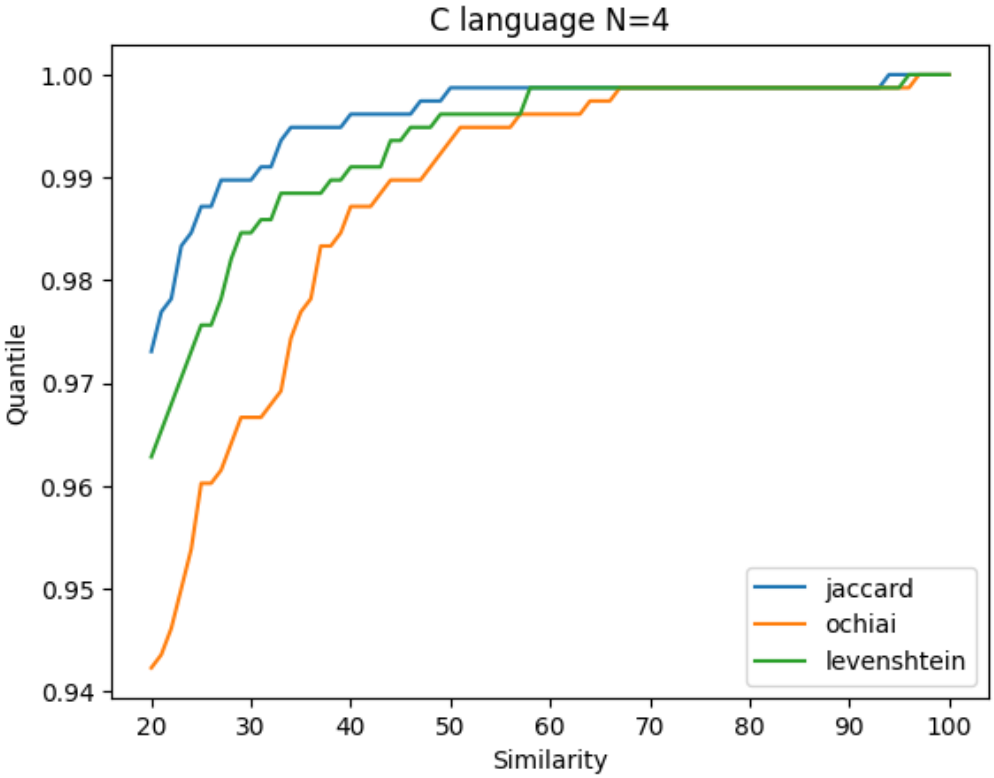


(Рисунок 3.6) Распределение коэффициентов сходства при длине N-граммы равной 4 (Архив 1).

Для наглядности на Рисунке 3.7 изображена диаграмма, на которой отображаются квантили для сходства от 10%, а на Рисунке 3.8 — от 20%



(Рисунок 3.7) Распределение коэффициентов сходства выше 10% при длине N-граммы равной 4 (Архив 1).



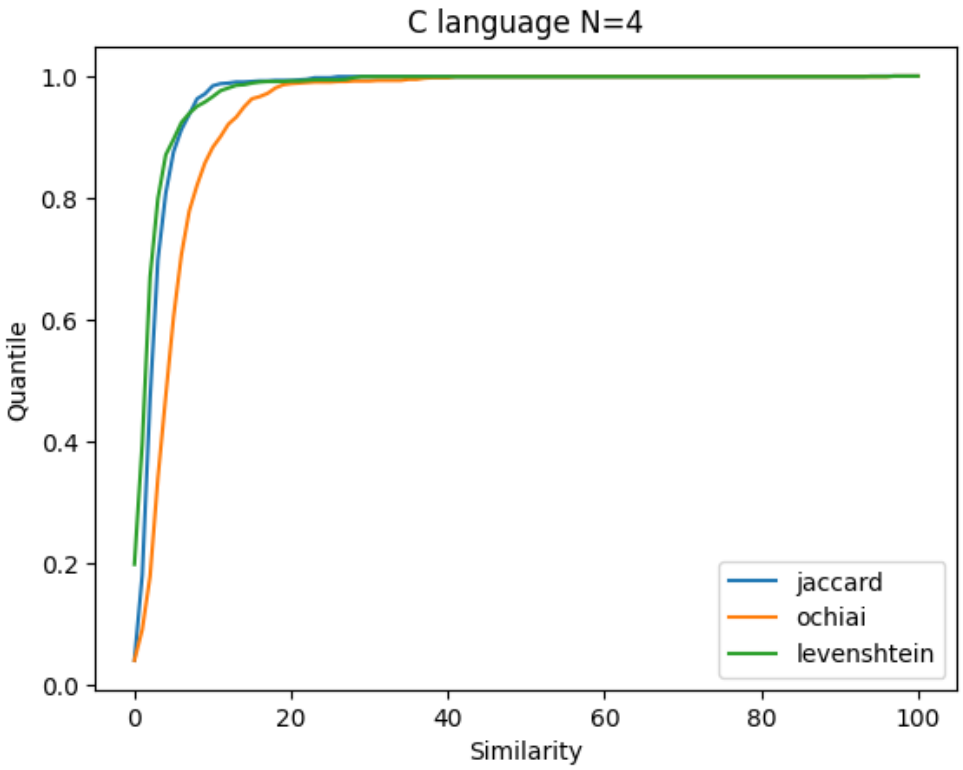
(Рисунок 3.8) Распределение коэффициентов сходства выше 20% при длине N-граммы равной 4 (Архив 1).

Архив 1 состоит из 45 программ. Следовательно, имеем 1035 пар программ, которые сравниваются. Программы в каждой паре сравниваем методом шинглов при длине шингла N=4 и производим подсчет метрик: расстояние Жаккара, коэффициент Отиаи и расстояние Левенштейна. Функция распределения сходства программ для трех метрик изображена на Рисунке 3.6.

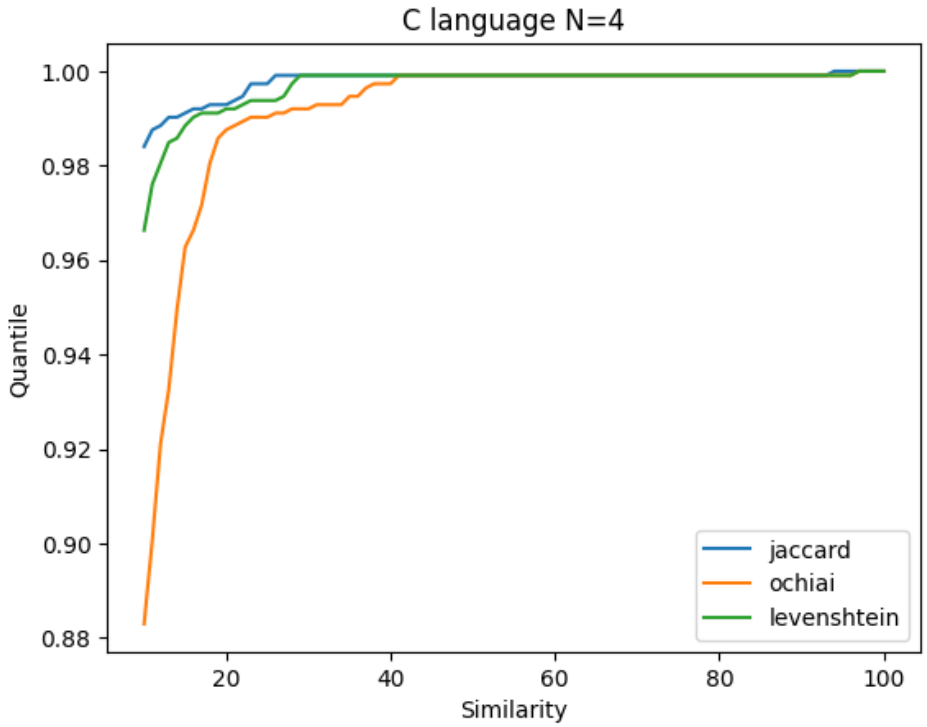
33% пар программ имеют сходство выше 10% для коэффициента Отиаи. Для расстояния Левенштейна таких пар 10%, а для расстояния Жаккара — 7,5% (Рисунок 3.7).

Чуть меньше 5% пар программ имеют сходство выше 20% для коэффициента Отиаи. Для расстояния Левенштейна таких пар чуть меньше 4%, и для расстояния Жаккара таких пар чуть меньше 3% (Рисунок 3.8).

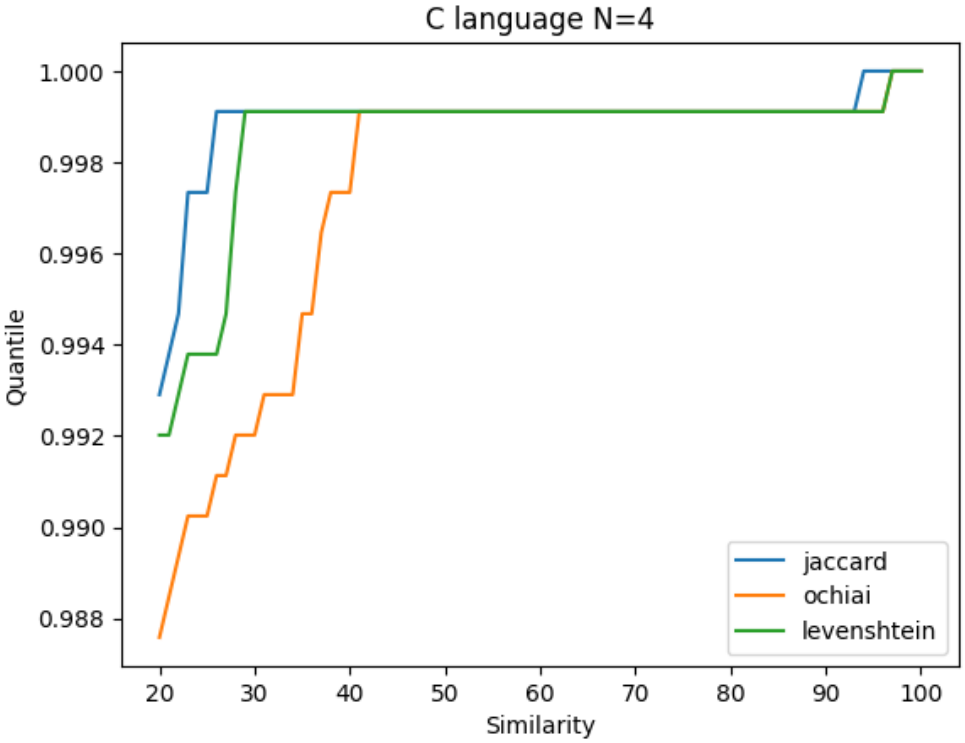
Рассмотрим диаграмму на Рисунке 3.9. Она отображает квантили распределения сходства студенческих программ на языке C из Архива 2 при подсчете расстояния Жаккара, коэффициента Отиаи и расстояния Левенштейна.



(Рисунок 3.9) Распределение коэффициентов сходства при длине N-граммы равной 4 (Архив 2).



(Рисунок 3.10) Распределение коэффициентов сходства выше 10% при длине N-граммы равной 4 (Архив 2).



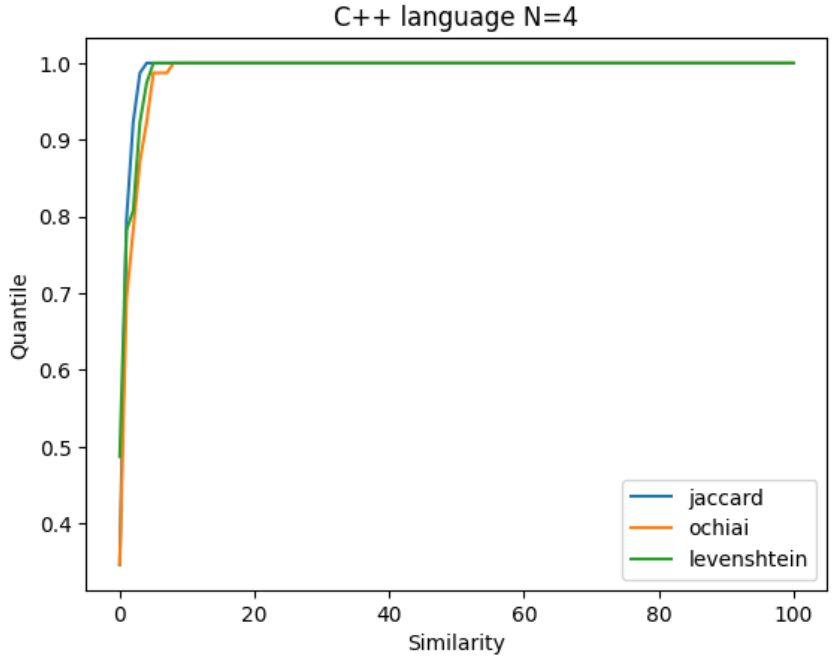
(Рисунок 3.11) Распределение коэффициентов сходства выше 20% при длине N-граммы равной 4 (Архив 2).

Архив 2 состоит из 48 программ. Следовательно, имеем 1128 пар программ, которые сравниваются. Программы в каждой паре сравниваем методом шинглов при длине шингла N=4 и производим подсчет метрик: расстояние Жаккара, коэффициент Отиаи и расстояние Левенштейна. Функция распределения сходства программ для трех метрик изображена на Рисунке 3.9.

12% пар программ имеют сходство выше 10% для коэффициента Отиаи. для расстояния Левенштейна таких пар чуть меньше 4%, а для расстояния Жаккара их немного менее 2% (Рисунок 3.10).

1,2% пар программ имеют сходство выше 20% для коэффициента Отиаи. Для расстояния Левенштейна и расстояния Жаккара таких пар примерно 0,7% (Рисунок 3.11).

На Рисунке 3.12 изображена диаграмма, на которой отображаются квантили распределения сходства студенческих программ на языке C++ из Архива 3.



(Рисунок 3.12) Сходство студенческих работ на языке C++ при длине N-граммы равной 4 (Архива 3).

В Архиве 3 находится 13 программ. Следовательно, 78 пар программ, которые сравниваются. По метрике коэффициент Отиаи все пары программ имеют сходство меньше 8%, по метрике расстояние Жаккара — меньше 5%, по метрике расстояние Левенштейна — меньше 4%.

Упорядочим метрики по убыванию показателя сходства для одних и тех же сравнений:

1. коэффициент Отиаи;
2. расстояние Левенштейна;
3. расстояние Жаккара.

Изначально преподаватели, предоставившие студенческие работы, не делали анализ на плагиат. Следовательно, такой архив не является полноценным датасетом для проверки эффективности алгоритма.

Когда система выдавала высокий процент сходства (выше 30%), преподаватели, которые предоставили студенческие программы для тестирования, считали, что при личной проверке программы в этой паре являются заимствованными друг у друга. Однако преподаватели не были полностью уверены, что сходства до 15% являются плагиатом. Проверяющим не стоит тратить время проверку плагиата в случае, когда сходство программ низкое. Например, для языка C менее 15%. Стоит учитывать также, что в зависимости от языка программирования меняется и процент сходства. Чем более низкоуровневый язык, тем сходство выше.

Можно сделать вывод: для одного и того же языка программирования, для одного и того же параметра длины N-грамма имеем разные показатели сходства для различных архивов. То есть показатели сходства зависят не только от языка программирования и параметров антиплагиата, но и от задачи, решение которой реализуют программы внутри архива. Таким образом, человек, который проверяет наборы программ на плагиат, должен самостоятельно настраивать порог процента сходства, считающегося критичным. Этот параметр не стоит указывать ниже 15%. Проверяющий должен провести анализ результатов проверки и в случае необходимости провести дополнительную проверку, чтобы убедиться в наличии плагиата.

Чем не является данная система? Это не система для полностью автоматического обнаружения плагиата. Плагиат - это утверждение о том, что кто-то намеренно скопировал код без указания авторства, и хотя система автоматически обнаруживает сходство программ, у неё нет способа узнать, почему коды похожи. Человек по-прежнему должен самостоятельно рассмотреть те части кода, которые выделяет система проверки, и принять решение о том, есть ли плагиат или нет. Преимущество системы заключается в том, что она экономит преподавательскому составу время, указывая на те студенческие работы, которые заслуживают более детального изучения. Однако, окончательное признание пары работ достоверным плагиатом должно быть проверено и надежно доказано.

4 Запуск модуля и описание формата вывода

Описание системы:

1. REST API архитектура приложения.
2. Реализована система на языке Python с использованием фреймворка flask.
3. Система разворачивается в Docker контейнере на 5000-м порту и слушает внешний 80-й порт.
4. Интерфейс системы — single page application.
5. Возможности интерфейса: загрузка файлов, настройка параметров проверки, просмотр результатов.

По адресу [23] находится инструкция по установке и запуску программы.

Порядок действий для запуска серверной части:

1. Загрузить на сервер, где будет развернут контейнер, исходный код сервиса ранжирования сходства студенческих программ. Также контейнер может быть запущен локально для персональной работы.
2. Запустить контейнер с сервисом, выполнив команду: ***make build && make run***
3. После того, как контейнер стартовал, пользователь, зная IP или DNS адрес сервера, может начать работу с сервисом.

Этапы работы пользователя с сервисом:

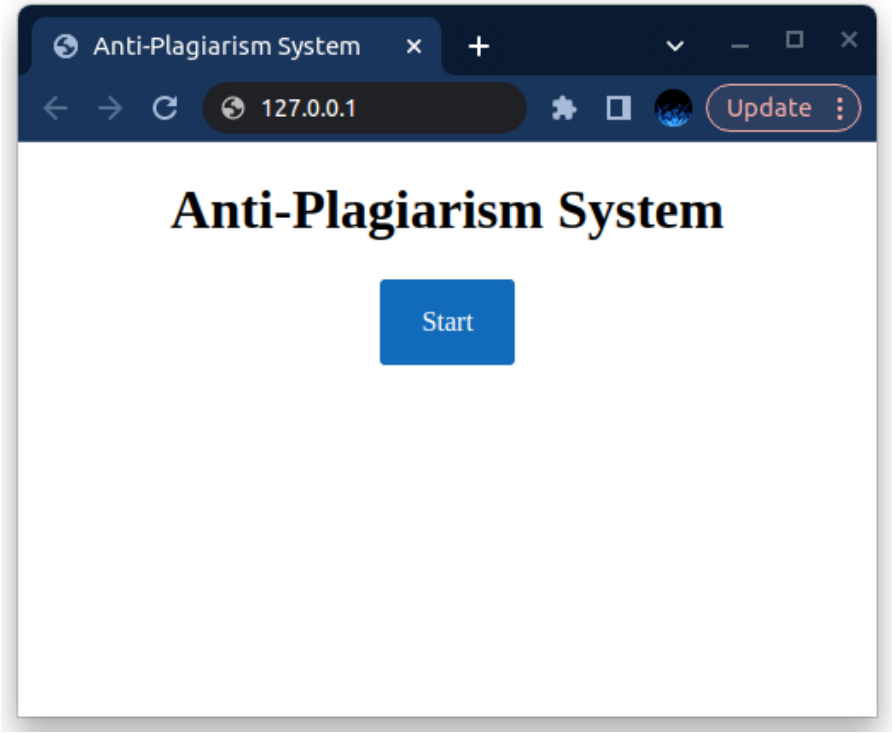
1. Загрузка пользователем архива с кодом, настройка параметров и запуск анализа.
2. Получение вычисленных результатов.

Функционал сервиса:

1. Получение архива программ от пользователя и разархивирование.
2. Определение языка программирования по расширению имени файлов.
3. Очистка кода от комментариев и разбиение на токены при помощи наборов регулярных выражений.
4. Объединение токенов в N-граммы и вычисление их контрольных сумм.
5. Попарное сравнение полученных наборов контрольных сумм с помощью метрик.
6. Вывод полученных результатов пользователю.

Web-интерфейс:

По IP или DNS адресу сервера (в случае локального запуска контейнера это будет ***localhost/***) имеется окно с кнопкой для перехода на главную функциональную страницу сервера (Рисунок 4.1).



(Рисунок 4.1) Главная страница детектора.

По пути ресурса ***/file-upload*** на сервере пользователь попадает в основное рабочее окно, где может загружать архив с программами для поиска сходства и настраивать параметры системы (Рисунок 4.2).

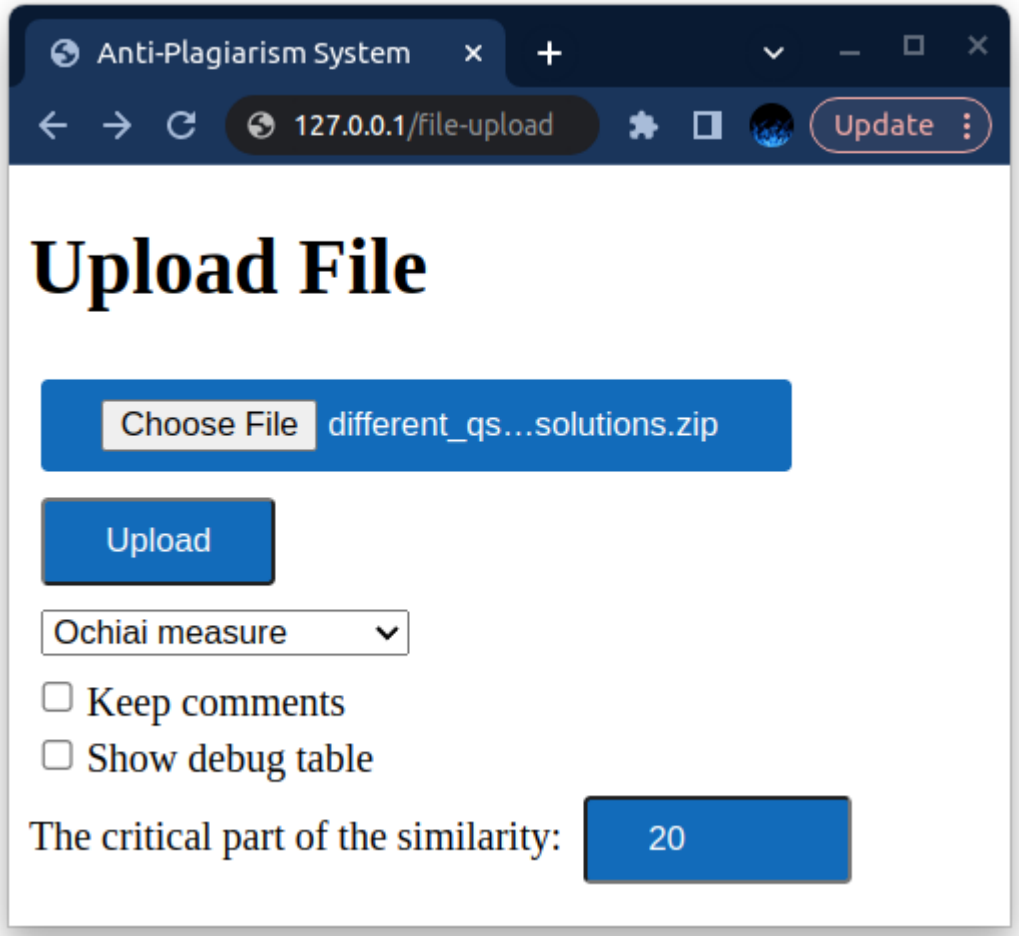
Доступные пользователю параметры, позволяющие управлять функциональностью системы:

- Выбор метрики сравнения программ внутри архива:
    - Коэффициент Отиаи выбран по умолчанию, так как ожидаются более высокие коэффициенты сходства по сравнению с расстоянием Жаккара.
    - Расстояние Жаккара — менее чувствительная метрика. Ею полезно пользоваться, если нужно увидеть только пары программ с высоким сходством.

- Расстояние Левенштейна высчитывается дольше, чем в случае коэффициента Отиаи и расстояния Жаккара, но в определенных случаях результат может быть более точным.

- Возможность не убирать комментарии при сравнении.
- Выводить отладочную таблицу, отображающую сравнение всех пар программ.
- Выбор порога процента сходства, считающегося критичным.

Кнопка ***Upload*** отправляет загруженный архив на обработку.

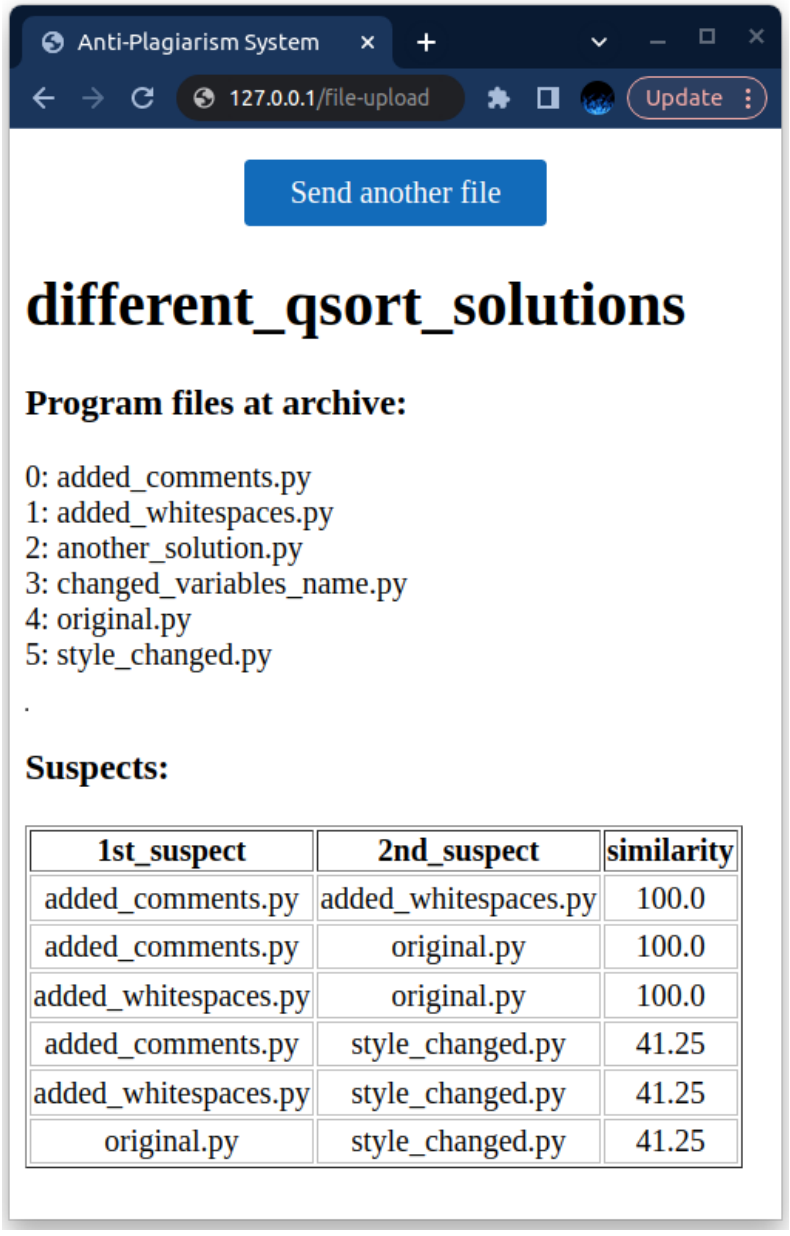


(Рисунок 4.2) Страница загрузки архива и выбор параметров.

После отправки архива на сервер пользователь попадает по тому же пути ресурса ***/file-upload*** на сервере на страницу вывода результатов работы системы (Рисунок 4.3). Кнопка ***Send another file*** возвращает на страницу с отправкой архива. На том же рисунке приведен результат ранжирования по сходству программ из тестового набора, описанного в Главе 3.

Программа выводит:

- Название загруженного архива (***different_qsort_solutions***).
- Названия обнаруженных программ в архиве с их номерами в лексикографическом порядке (***Program files at archive***).
- Схожие пары программ, отсортированные по убыванию сходства. Для каждой пары выводятся их названия и показатель сходства между ними (***Suspects (first suspect, second suspect and similarity)***).



(Рисунок 4.3) Вывод результатов работы детектора по заданным параметрам.

Полученные оценки и есть система ранжирования. Она полезна для мониторинга относительной степени совпадения между различными парами небольших студенческих программ. Наглядно видно, какие пары программ имеют высокое текстовое сходство и требуют экспертной оценки.

ЗАКЛЮЧЕНИЕ

- В данной работе выполнен обзор существующих программных средств для обнаружения плагиата, проанализированы их преимущества и недостатки. Исследованы различные методы обнаружения плагиата в коде.
- Использован метод шинглов для обнаружения сходства двух сравниваемых программ, в котором проводится предобработка, выбор метрик и определение лучшего разбиения на N-граммы.
- Реализована система ранжирования пар небольших программ, написанных на одном языке программирования студентами младших курсов. Она показывает «кандидатов», подозреваемых в плагиате, на которых стоит обратить внимание проверяющему преподавателю.
- Работоспособность системы подтверждена результатом проверки на тестовом датасете. Показана эффективность такой системы проверки, что подтверждает возможность её применения в реальных условиях.

Программа написана на языке Python.

Код программы выложен на GitHub [23].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] Плагиат [Электронный ресурс]. — Электрон. дан. — URL: https://gufo.me/dict/bes/ПЛАГИАТ (дата обращения: 2.05.2023)

[2] Hosam, E., Hadhoud, M., Atiya, A. et al. (2022) Classification feature sets for source code plagiarism detection in Java. J. Eng. Appl. Sci. 69, 102

[3] Плагиат: что это такое простыми словами [Электронный ресурс] — Электрон. дан. — URL: https://plagiatfree.ru/stati/plagiat-chto-eto-takoe-prostymi-slovami/ (дата обращения 19.05.2022)

[4] Антиплагиат [Электронный ресурс]. — Электрон. дан. — URL: https://www.antiplagiat.ru/ (дата обращения: 19.05.2022)

[5] Проверка уникальности eTXT [Электронный ресурс]. — Электрон. дан. — URL: https://www.etxt.ru/ (дата обращения 19.05.2022)

[6] Сервис проверки текста на уникальность и биржа контента TEXT [Электронный ресурс]. — Электрон. дан. — URL: https://text.ru/ (дата обращения: 19.05.2022)

[7] Анти-антиплагиат [Электронный ресурс]. — Электрон. дан. — URL: https://анти-антиплагиат.рф/blog/samyj-luchshij-horoshij-antiplagiat (дата обращения 19.05.2022)

[8] Романов А. С. Методика проверки однородности текста и выявления плагиата на основе метода опорных векторов и фильтра быстрой корреляции / А. С. Романов, Р. В. Мещеряков, З. И. Резанова // Доклады ТУСУР. – 2014. – № 2(32). – С. 264–269.

[9] JPlag [Электронный ресурс]. — Электрон. дан. — URL: https://github.com/jplag/JPlag (дата обращения 19.05.2022)

[10] MOSS [Электронный ресурс]. — Электрон. дан. — URL: https://theory.stanford.edu/~aiken/moss/ (дата обращения 19.05.2022)

[11] Copyleaks [Электронный ресурс]. — Электрон. дан. — URL: https://copyleaks.com/ (дата обращения 16.04.2023)

[12] Обзор автоматических детекторов плагиата в программах [Электронный ресурс]. — Электрон. дан. — URL: https://studylib.ru/doc/176375/obzor-avt (дата обращения 21.03.2023)

[13] Евтифеева О.А., Красс А.Л., Лакунин М.А., Лысенко Е.А., Счастливцев Р.Р. Анализ алгоритмов поиска плагиата в исходных кодах программ [Электронный ресурс]. — Электрон. дан. — URL: https://cyberleninka.ru/article/n/analiz-algoritmov-poiska-plagiata-v-ishodnyh-kodah-programm/viewer (дата обращения 11.05.2023)

[14] Saul Schleimer, Daniel S. Wilkerson, Alex Aiken, Winnowing: local algorithms for document fingerprinting, Proceedings of the 2003 ACM SIGMOD international conference on Management of data, June 09-12, 2003, San Diego, California.

[15] Александр Зоркин. Антиплагиат исходного кода: гибридный подход с использованием парсера ANTLR [Электронный ресурс]. — Электрон. дан. — URL: https://habr.com/ru/post/583882/ (дата обращения 19.05.2022)

[16] Расстояние Левенштейна [Электронный ресурс]. — Электрон. дан. — URL: https://ru.wikipedia.org/wiki/Расстояние_Левенштейна (дата обращения: 11.05.2023)

[17] И. А. Посов, В. Е. Допира. Методы поиска плагиата в кодах программ [Электронный ресурс]. — Электрон. дан. — URL: https://izv.etu.ru/assets/files/izvestiya-6_2019_p061-066.pdf (дата обращения 23.03.2023)

[18] Коэффициент Отиаи [Электронный ресурс]. — Электрон. дан. — URL: https://ru.wikipedia.org/wiki/Коэффициент_Отиаи (дата обращения: 11.05.2023)

[19] Geoff Whale (1990), Software Metrics and Plagiarism Detection, Department of Computer Science, University of New South Wales, Australia

[20] Binascii [Электронный ресурс]. — Электрон. дан. — URL: https://docs.python.org/3/library/binascii.html (дата обращения: 8.05.2023)

[21] Python 3 docs [Электронный ресурс]. — Электрон. дан. — URL: https://docs.python.org/3/ (дата обращения: 8.05.2023)

[22] UnPlag [Электронный ресурс]. — Электрон. дан. — URL: https://scriptographers.github.io/UnPlag/#getting-started (дата обращения: 9.05.2023)

[23] Разработанная программа [Электронный ресурс]. — Электрон. дан. — URL: https://github.com/KoSTYAa/final_qualifying_work (дата обращения: 11.05.2023)