# ПОИСК ПЛАГИАТА В ТЕКСТАХ ИСХОДНЫХ КОДОВ ПРОГРАММ

**Р. Ю. Роуба**

*Белорусский государственный университет, г. Минск;*
*fpm.rouba@bsu.by;*
*науч. рук. – В. Ю. Сакович, ст. преп.*

В данной статье рассматривается решение проблемы эффективного поиска плагиата в тексте программ с использованием одной из версий алгоритма отпечатков. Помимо конкретной реализации в виде функционального api и пользовательского интерфейса, для целей демонстрации в данной работе обуславливается эффективность выбранного подхода к решению поставленной задачи, а также рассматривается возможность обобщения применённого метода к схожему классу задач. Все проведённые теоретические рассуждения подкреплены данными тестирования на реальных примерах.

***Ключевые слова:*** поиск плагиата, алгоритм отпечатков, методы трансляции.

## ВВЕДЕНИЕ

При рассмотрении работы различных систем поиска плагиата отдельным классом задач является поиск плагиата в текстах программ. Основной спецификой является строгость задающих языки программирования грамматик, а также наличия явной и последовательной логики их интерпретации. В связи с этим возникают различные специализированные подходы как к распознаванию плагиата, так и специфические способы его сокрытия. Соответственно, одним из основных критериев оценки эффективности алгоритма будет являться его способность верно находить не только банальное копирование, но и попытки маскировки такового.

## АЛГОРИТМ ОТПЕЧАТКОВ

В данной работе рассматривается алгоритм, базовой идеей которого является выбор из всего файла исходного кода некоторых наиболее полно его характеризующих фрагментов – отпечатков, дабы в дальнейшем производить анализ заимствований, исходя из этих конкретных значений. Такой подход позволяет быстро сравнивать файлы по модели «один ко многим», что и требуется в большинстве случаев, а также может существенно сократить объём требуемой памяти в случае особого вида хранения отпечатков. Наиболее часто используемый подход, что не стало исключением в данной реализации,

заключается в хранении хэшированного представления отпечатка, а также его смещения относительно начала файла для возможности отображения информации конечному пользователю.

Для отбора отпечатков существует несколько подходов, однако наиболее точным из всех является алгоритм со скользящим окном [1]. Суть его заключается в выборе всего одного хэш-значения из нескольких входящих в окно, которое «пробегает» по всему файлу с единичным шагом. Регулируя размер окна, можно задавать минимальный участок кода, считающийся плагиатом, а также гарантировать выбор в качестве отпечатка всех участков кода более некоторой фиксированной длины. Параметры размера окна и единиц для хэширования были подобраны, исходя из тестирования на реальных данных. Также было проверено утверждение о большей точности данного алгоритма с использованием хэширующих функций из семейства Карпа-Рабина [2], но из-за особенностей реализации предварительной обработки данный подход не оправдал себя.

## ПРЕДВАРИТЕЛЬНАЯ ОБРАБОТКА

Для работы самого алгоритма от момента хэширования и до момента сравнения хэш-значений не имеет никакого значения вид поступающего на вход текста. Однако как банальная логика, так и результаты тестирования показали, что от первичного преобразования текста исходного кода результат работы может очень сильно варьироваться. Поэтому в финальном варианте был представлен наиболее интересный как с теоретической, так и практической точки зрения результат – трансляция исходного языка программирования в некоторое промежуточное и обобщённое представление. Эффективность такого подхода целиком и полностью зависит от полноты описания всех возможных правил перевода, а также широты нового алфавита, что является достаточно большой темой для отдельного исследования. Явным же плюсом можно указать независимость сравнения прошедших через такую обработку файлов с кодом на разных языках, например на Java и C++, ведь оба варианта приводятся к единому представлению.

## РЕЗУЛЬТАТЫ

Итак, опишем полученный результат. Разработанное приложение состоит из четырёх независимых модулей, реализующих предварительную обработку файла (приведение исходного кода к обобщённому виду), непосредственное хэширование и выбор

отпечатков, сравнение хэш-значений с предоставлением дополнительной информации, графического интерфейса для демонстрации работы. Благодаря набору решений приложение успешно фиксирует не только наличие прямого копирования, но также перестановки блоков кода (выбор отпечатков с помощью окна), изменение имён идентификаторов (трансляция к обобщённому представлению), дробление на функции (трансляция + окно) и некоторые другие, а также любую из комбинаций вышеперечисленных попыток сокрытия факта плагиата.

Тестирование корректности работы приложения, а также подбор оптимальных параметров производился на наборе решений контрольных работ студентов прошлых лет, включающих в себя более 100 уникальных архивов с файлами на языке программирования Java. Разнообразие стилей написания, а также явные попытки взаимоплагиата позволили улучшить изначальную версию транслятора до достаточно точной, но не излишне строгой версии, а также подобрать оптимальные параметры для размеров окна. Итоговая версия работает с инструкциями, превращая по 3 инструкции в одно хэш-значение, а размер окна составляет 5 инструкций. Таким образом, незначительно малые участки кода длиной до трех инструкций в принципе не рассматриваются подозрительными на плагиат, а выбор отпечатка происходит не реже чем раз в 5 строк кода, что позволяет отлавливать разбиение даже на относительно небольшие функции.

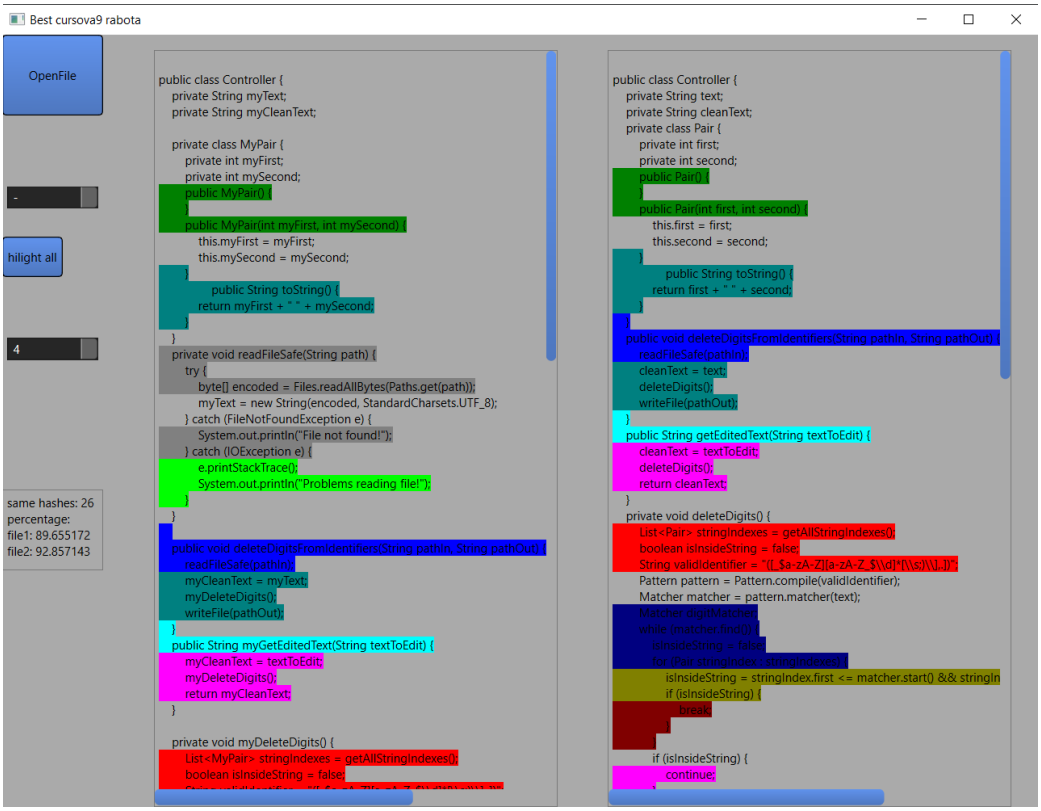
Графический интерфейс, представленный ниже, снабжён всем необходимым для демонстрации функционалом: возможностью открывать файл для проверки, выбором отсортированного в порядке убывания количества совпадений файла из базы для сравнения, возможностью визуализировать как отдельные совпадения, так и сразу все.

## ЗАКЛЮЧЕНИЕ

В целом, разработанное приложение является полностью портативным и встраиваемым в любую существующую систему (при наличии совпадающего api), единственным направлением для развития является модуль для трансляции кодов, который можно независимо дополнять правилами для различных языков, а также вносить изменения в имеющиеся для улучшения их работы. Система является весьма легковесной и быстрой, и, соответственно, может быть применена в реальных условиях, и в то же время может представлять интерес с

научной точки зрения, а именно с позиции построения и влияния транслятора на точность результата.

Общий вид приложения представлен на рисунке.




Пример работы алгоритма с графическим интерфейсом

**Библиографически ессылки**

1. *Aiken A.* Winnowing: local algorithms for document finger-printing / *A. Aiken, S. Schleimer, D. Wikerson* // Proceedings of ACMSIGMOD Int. Conference on Management of Data. – San Diego, 2003. – P. 76-85.
2. *Wise, M.J.* String similarity via greedy string tiling and running Karp-Rabin matching / *M. J. Wise* // Dept. of CS, University of Sydney. – 1993. – P. 34-47.