УДК 004

Часов Е. А., старший преподаватель
кафедры информатики и вычислительной техники
ФГБОУ ВО ПГУТИ, Россия, г. Самара
Марина М. А., старший преподаватель
кафедры информатики и вычислительной техники
ФГБОУ ВО ПГУТИ, Россия, г. Самара
Киптенко А. В., студент гр. ПО-73
ФГБОУ ВО ПГУТИ, Россия, г. Самара
Слеппев Д. А., студент гр. ИСТ-62
ФГБОУ ВО ПГУТИ, Россия, г. Самара

РАЗРАБОТКА МЕТОДОВ ПОИСКА ЗАИМСТВОВАНИЙ В ИСХОДНОМ КОДЕ НА ЯЗЫКАХ ПРОГРАММИРОВАНИЯ

Аннотация: В данной статье описывается анализ методов поиска заимствований в исходном коде, их достоинства, недостатки, поиск оптимального подхода к решению задачи, сравнение с задачей поиска заимствований в тексте на естественном языке.

Ключевые слова: Java, плагиат, заимствование, синтаксический анализ, графы, семантический анализ.

Chasov E. A.,
Lecturer of Department
of Informatics and Computer Engineering
Povolzhskiy State University
of Telecommunications and Informatics
Russia, Samara
Marina M. A.,
Lecturer of Department
of Informatics and Computer Engineering
Povolzhskiy State University

of Telecommunications and Informatics
Russia, Samara
Kiptenko A. V.
Student
Povolzhskiy State University
of Telecommunications and Informatics
Russia, Samara
Slepnev D. A.
Student
Povolzhskiy State University
of Telecommunications and Informatics
Russia, Samara

DEVELOPMENT OF METHODS FOR PLAGIARISM SEARCHING IN THE SOURCE CODE IN PROGRAMMING LANGUAGES

Abstract: In this article analyzes of methods for searching of plagiarism in source code, their advantages and disadvantages, searching of the most optimal approach for the problem solution, comparing with the problem of searching plagiarism in the text in the natural language are described.

Keywords: Java, plagiarism, syntax analyze, graphs, semantic analyze.

Обычно в вопросах поиска заимствований в тексте подразумевается, что он осуществляется в текстах на естественном языке: поиск плагиата сегодня является актуальным вопросом во многих областях: проверка студенческих курсовых работ, выпускных квалификационных работ, проверка диссертаций на соискание степени кандидата или доктора наук и др. Поиск плагиата в текстах может осуществляться также и в промышленности, если это имеет экономическое и/или правовое обоснование.

В таких задачах реализация поиска заимствований осуществляется при помощи специально разработанных алгоритмов. Проверка

осуществляется по самым разным критериям, с учетом попыток обхода. Кратко рассмотрим методы поиска заимствований и попыток обхода систем проверки.

Самый простой метод проверки текста на заимствования – построчное сравнение. Каждая строка целевого текста сравнивается с со строками файлов, находящимися в индексированной хранящейся базе данных. Этот метод не требует сложных алгоритмических решений и имеет очевидные недостатки: малейшие изменения в строке дадут ложный результат сравнения.

С течением времени алгоритм поиска дополнился продвинутыми решениями: из текста стали вычленять значимые слова, термины и проводить сравнение между ними. При таком решении наличие в тексте различных предлогов, междометий и других незначительных для содержимого частей речи оказалось безразличным для алгоритма поиска. Далее для сравнения отдельных цитат стали применять алгоритмы построения и сравнения суффиксных деревьев, строящихся для отдельно взятых значимых и терминов – такой подход позволил проводить сравнение текста, игнорируя измененные окончания в словах.

В большинстве случаев если в алгоритмах поиска не продуман вопрос проверки на попытки обхода системы, то многие существующие методы возможно обмануть: использовать вместо классического пробельного символа символ с другим кодом, заменять знак абзаца на знак разрыва строки, использовать в тексте символы другого алфавита с одинаковым начертанием. Эти методы были популярны, но сейчас существующие системы способны это обнаружить.

Попытки обхода тоже развивались с течением времени: в документ стали внедрять графические объекты с текстовым содержанием, чтобы процент заимствования снижался, бессмысленный по смыслу текст внедрялся в нечитаемые текстовым редактором области документа, но

читаемые системой проверки; части текста заменялись на изображение-скриншот этой части. С течением времени и такие попытки обхода стали обнаруживаться и системы проверки перешли в новый этап развития, стал применяться метод стилометрии.

Стилометрия или изучение языковых стилей — это статистический метод для выявления авторства документов и проверки на плагиат. Строятся стилометрические модели для различных фрагментов текста, отрывков, которые стилистически отличаются от других. И путём сравнения моделей можно обнаружить плагиат.

При этом в продвинутых системах проверки может применяться латентно-семантический анализ для определения смысла содержимого, и на его основе выявлять попытки пересказа текста.

Попытки обхода таких методов уже могут заранее обернуться неудачей, т.к. автоматическое преобразование текста вряд ли принесет результат без человеческого вмешательства.

Описанные выше методы, разумеется, как было сказано, применимы для текстов на естественном языке. Сравнение цитат, построение суффиксных деревьев, попытки обхода систем проверки – все это абсолютно неприменимо для задач проверки на плагиат исходного кода на любом языке программирования: любой из них является формальным языком, и задача обнаружения плагиата лежит в плоскости формального подхода к задаче, построению математической модели и определению методов проверки кода.

Кратко опишем основные отличия в сравнении текста на естественном языке, от кода на языке программирования, т.е. текста на формальном языке. Сравнение текста программы на языке программирования, в большинстве случаев, должно игнорировать отступы, пробельные символы, пустые строки и комментарии. При совпадении определенных участков кода сравнение должно в общем случае

игнорировать имена идентификаторов (переменных, классов, интерфейсов, пакетов и др.).

Отбросим способы, с помощью которых нельзя достоверно определить плагиат в исходном коде. Для выявления совершенно не подходят способы:

Полное сравнение целиком изначально не окажется полезным: изменение даже одного символа даст ложный результат сравнения. Сработает только в случае полного копирования кода без изменений.

Построчное сравнение не даст достоверного результата, т.к. исходный код в большинстве случаев изменение структуры строк не влияет на изменение логики приложения, равно как игнорируются пустые строки, а также строки, содержащие комментарии. Также построчное сравнение осложняется тем, что в случае попытки сокрытия плагиата путем добавления бесполезных синтаксических конструкций, процент заимствования будет снижен, как в случае добавления бессмысленного или оригинального текста в документ на естественном языке.

Вышеописанное также позволяет сделать вывод, что посимвольное сравнение также не окажется эффективным, поскольку требуются иные методы работы с исходным кодом.

Рассмотрим и проанализируем методы, при помощи которых можно определить плагиат исходного кода. Разумеется, первым и необходимым условием является формирование индекса и базы данных, с которой будет сравниваться проверяемый исходный код.

Предлагаемый подход охарактеризован тем, что при работе с кодом сперва следует учитывать, что его анализ должен производиться с игнорированием пустых строк, пробелов и комментариев – подход, с которым работает компилятор, осуществляя преобразование текста на формальном языке в машинные команды. Ввиду этого анализ кода следует проводить аналогичного этапам работы компилятора: лексическому,

синтаксическому и семантическому анализу. Для этого следует, проведя лексический анализ, провести синтаксический и построить синтаксическое дерево кода, семантически аналогичное тому, что строит компилятор, но использоваться оно будет по-другому.

Для его построения следует разработать формальную грамматику языка, на котором написан код. Для этого можно использовать существующие генераторы парсеров на основе грамматик, например, ANTLR. Рассмотрим небольшой пример. Возьмем модельный псевдо-язык программирования и составим для него грамматику. Приведем ниже ее часть:

```
grammar Model;
parse : block EOF ;
block : stat* ;
stat : assignment
| if_stat
| while_stat
| log
| OTHER {System.err.println("unknown char: " + $OTHER.text);} ;
expr : expr POW<assoc=right> expr #powExpr
| MINUS expr #unaryMinusExpr
| NOT expr #notExpr
| expr op=(MULT | DIV | MOD) expr #multiplicationExpr
| expr op=(PLUS | MINUS) expr #additiveExpr
| expr op=(LTEQ | GTEQ | LT | GT) expr #relationalExpr
| expr op=(EQ | NEQ) expr #equalityExpr
| expr AND expr #andExpr
| expr OR expr #orExpr
| atom #atomExpr ;
SCOL : ';';
```

ASSIGN : '=';

OPAR : '(';

CPAR : ')';

OBRACE : '{';

CBRACE : '}';

Данная грамматика содержит правила для лексического анализатора, начинающиеся с прописной буквы, для удобства все эти правила написаны строчными буквами, и правила, начинающиеся со строчной буквы – для синтаксического анализатора. На основе этих правил ANTLR генерирует классы и интерфейсы для лексического и синтаксического анализа. Грамматика позволяет написать код следующего вида:

```
a = 10;
b = 100;
if (a>b) {
a = a+b;
}
```

Для работы со сгенерированными классами требуется реализовать некоторые алгоритмы обхода синтаксического дерева и прочей информации, полученной в результате обработки исходного кода парсером.

В общем виде код подается на вход парсеру, который выделяет из кода лексемы, делает вывод о корректности кода с точки зрения синтаксиса, строит синтаксическое дерево, которое необходимо нам для работы системы проверки кода на объем заимствований. Основной структурой, необходимой для этого, является синтаксическое дерево. Оно позволяет сравнивать части программы или ее целиком и делать выводы о фактах заимствования.

Дерево позволяет игнорировать имена идентификаторов, сравнивая непосредственно логику приложения через описанные синтаксические

конструкции. При этом сравнение имен идентификаторов всё равно остается одним из факторов, позволяющих сделать вывод о заимствовании.

Оперирование деревом позволяет совершать обход по нему, сравнивания необходимые его части, это означает, что в хранимой БД исходного кода, на основе которой происходит анализ, должна также храниться информация об этапах анализа кода, либо должна получаться в режиме реального времени. В первом случае требуется большее место на диске для хранения, во втором – большие вычислительные ресурсы.

Синтаксическое дерево исходного кода в примере приведено на рисунке 1.

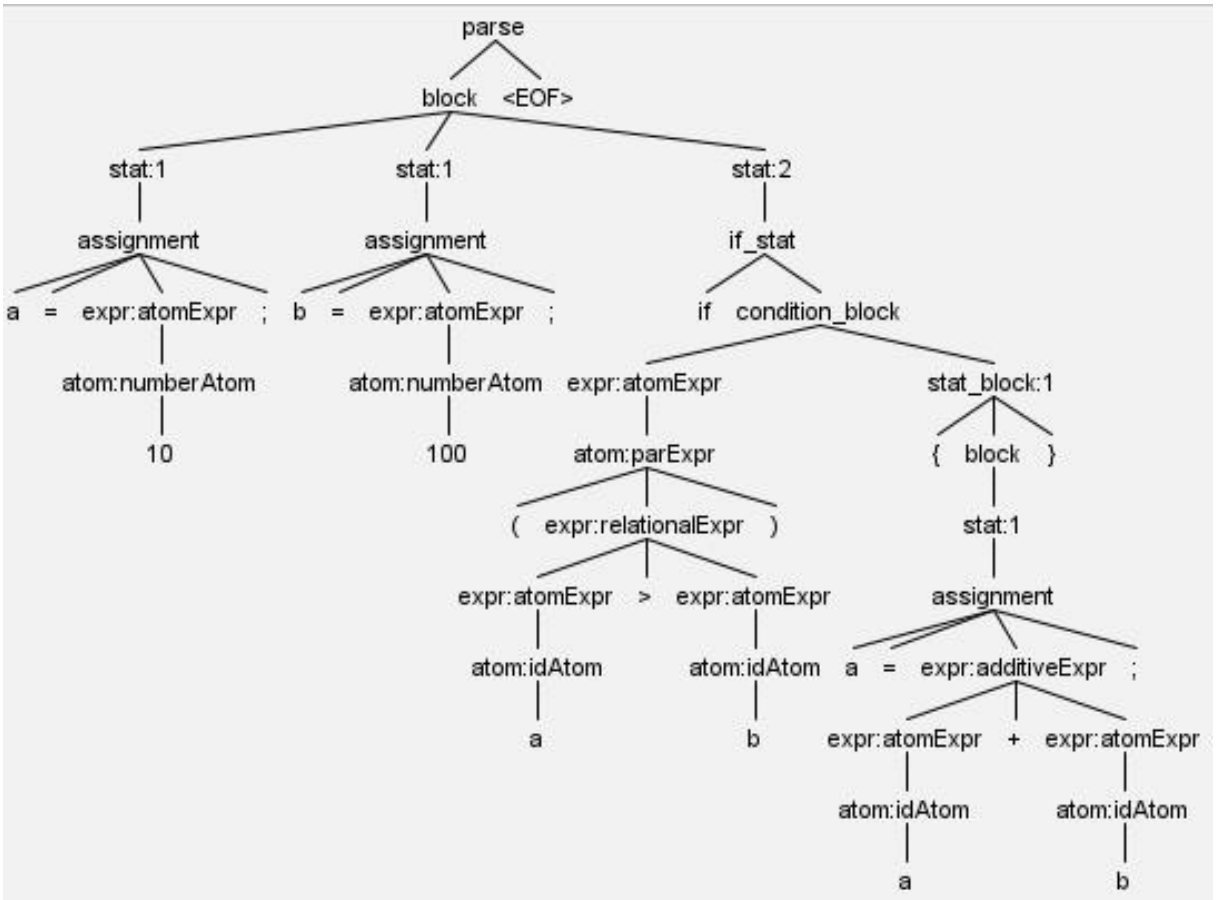

Рисунок 1. Синтаксическое дерево кода на модельном языке

Синтаксическое дерево не является достаточным условием для того, чтобы определить плагиат. Следующим этапом анализа плагиата предлагается создание дополнительной структуры данных, основанной на дереве. Суть заключается в следующем: на основе дерева проводится

анализ и выявление частей кода, в которых осуществляется вызов методов и функций. При этом должны отдельно учитываться вызовы пользовательских методов приложения, отдельно вызов методов стандартной библиотеки языка программирования, а также отдельно учитываться вызовы методов сторонних библиотек, которые являются зависимостями приложения.

На основе полученной информации должен строиться так называемый граф вызова методов, который позволяет получить дополнительную информацию о логике приложения. Аналогично такой же граф должен строиться и/или храниться на основе кода в индексируемой БД.

Анализ двух графов на идентичность позволит повысить точность определения плагиата. Прогнозируемый недостаток сравнения графов заключается в том, что бесполезный вызов методов в коде, например, вывод в консоль пустых строк, изменит граф методов, что снизит точность выводов при их сравнении. Чтобы избежать подобного, следует хранить дополнительный список таких методов, которые не влияют на логику работы приложения.

При этом сравнение графов при наличии таких методов всё равно следует реализовать таким образом, чтобы, проверяя графы на подобность, учитывать это и вычленять из графы те его части, которые являются значимыми.

Следующим этапом анализа является обработка и сравнение имен идентификаторов. Именование объектов в языках программирования, как правило, подвержено конвенциям программирования и стандартам разработки, но всё равно не лишено авторского отпечатка и особенностей, характерных для каждого программиста в отдельности. Стиль именования индивидуален и должен учитываться при анализе.

Отдельно следует отметить, что существуют некоторые решния, которые позволяют осуществлять проверку исходного кода на соответствие конвенциям, т.н. Check-style. Полнота и степень следования конвенциям позволит точнее определять авторство кода.

Стиль программиста влияет не только на именование объектов в исходном коде, но также отражается в целом на всей логике приложения. Стиль и логика написания кода разнятся в зависимости от того, кто является автором и его определение также влияет на точность распознавания плагиата. В этой задаче предполагается применить нейросетевые алгоритмы, однако это влечет за собой необходимость в обучении сети.

При этом определение стиля написания позволяет приблизиться к задаче определения смысла и семантики того кода, который подвергается анализу. На основе знания о том, для чего используется та или иная синтаксическая конструкция, для чего нужен тот или иной класс или метод (это выясняется на основе именования или документации), можно сделать вывод о смысле и цели работы описанного в коде алгоритма. Для этого можно также использовать нейросетевые алгоритмы с применением адаптированного латентно-семантического анализа. Знание о смысле и назначении алгоритма позволит нейросети сделать вывод не только о заимствовании и плагиате, но теоретически позволяет выявить логические ошибки, допущенные программистом.

Описанные методы обнаружения плагиата в исходном коде требуют тщательного и ручного тестирования на основе базы данных исходного кода с явным знанием о том, какой код является оригинальным, а какой был заимствован. Тем не менее, в случае успешной реализации и пройденном тестировании подобная система антиплагиата для исходного кода на разных языках программирования может быть успешна применена и использована в учебном процессе при работе со студентами, при

проверки выпускных квалификационных работ и диссертаций, при регистрации и проверке патентов и изобретений, а также способствовать защите интеллектуальных прав.

Использованные источники:

1. Эккель, Б. Философия Java. Библиотека программиста [Текст]/Б. Эккель – 4-е изд. – СПб.: Питер, 2009. – 640 с.

2. Шилдт, Г. Java. Полное руководство [Текст]/Г. Шилдт – 8-е изд.: пер. с англ. – М.: ООО «И.Д. Вильямс», 2012. – 1104 с.

3. Спецификация языка Java [Электронный ресурс]/2017.– Режим доступа: http://java.sun.com/docs/books/jls, свободный. – Загл. с экрана

4. Выявление плагиата [Электронный ресурс]/2019– Режим доступа: https://ru.wikipedia.org/wiki/Выявление_плагиата, свободный. – Загл. с экрана

1. Eckel, B. Thinking in Java [Text]/B. Eckel – 4-d ed. – MindView Inc, 2009. – 640 p.

2. Schildt, H. Java: The Complete Reference [Text]/H. Schildt – 9-th ed.:. – Osborn, Inc 2012. – 1104 p.

3. Javadoc [URL]/2019.– URL: http://java.sun.com/docs/books/jls

4. Plagiarism detection [URL]/2019– URL: https://ru.wikipedia.org/wiki/Выявление_плагиата/