Алгоритм выявления плагиата в исходном коде программ

П. А. Морозов, email: morozov_p.a@bk.ru

[1] Московский авиационный институт (национальный исследовательский университет)

Аннотация. *В данной работе рассматриваются основные типы заимствования программного кода, проводится анализ методов представления исходного кода и выявления заимствований. Предлагается алгоритм, основанный на токенизации и реализации алгоритмов Хескела и Хиршберга.*

Ключевые слова: *плагиат, программный код, способы представления программ, токены, алгоритм Хескела, алгоритм Хиршберга*.

Введение

В современном мире любое образовательное учреждение имеет дело с проблемой плагиата учащимися частей или полных работ по различным темам, что не только нарушает авторские права, но и негативно сказывается на качестве получаемого образования. Данную проблему решают различные сервисы проверки текстовых работ на плагиат.

Но самостоятельная деятельность учащихся не ограничивается только лишь текстовыми работами. Так, преподаватели учебных заведений, ведущих подготовку IT-специалистов, проводят обучение и контроль работы с программным кодом, проверка на плагиат которого становится затруднительной с помощью вышеуказанных сервисов. Учащийся может копировать лишь часть исходного кода, изменить имена переменных и функций, поменять расположение блоков, что визуально изменит код работы, но не скажется на ее выполнении.

Кроме того, некорректные заимствования программного кода являются серьезной проблемой и при разработке программных продуктов.

Таким образом, становится актуальна задача разработки алгоритма, обеспечивающего автоматическое выявление некорректных заимствований исходного кода программ на разных языках программирования. Метод должен позволять обнаруживать различные типы заимствования с достаточной точностью для его использования

как при проверке учебных работ, так и в дальнейшем для применения в области анализа программного кода программных продуктов.

1. Типы заимствования и обзор методов представления программ

Основными типами заимствования исходного кода программ являются [1]:

1. Тип T1 – код скопирован без каких-либо изменений, т.е. идентичен оригинальному.

2. Тип T2 – код скопирован с заменой идентификаторов (имен переменных, типов, функций, строковых литералов).

3. Тип T3 – может включать заимствования T2, а также изменение копированного кода путем добавления, изменения или удаления его фрагментов.

4. Тип T4 – копирование логики работы и функциональности. Синтаксически код может абсолютно отличаться от заимствованного. Данный тип крайне затруднителен для выявления, т.к. сложен в реализации и зачастую скопированный фрагмент не заимствован, а реализовывает одинаковую с оригиналом логику работы программы, т.е. фактически является заимствованием алгоритма, а не код программы.

Промежуточное представление программного текста (далее – представление) – это синтаксически и семантически эквивалентный исходному коду набор данных, над которым выполняется анализ [2]. Представление может быть выполнено в виде различных наборов данных:

1. Текстовое представление.

Исходный код программы рассматривается как последовательность строк. Данная модель производит поиск заимствований аналогично поиску плагиата в текстах, что позволяет обнаружить только заимствования типа T1. К положительным сторонам использования этого представления можно отнести простоту реализации и независимость от языка программирования.

2. Представление в виде абстрактного синтаксического дерева.

Поиск заимствований в коде на основе построения абстрактного синтаксического дерева (АСД) позволяет представить код в форме, не учитывающей информацию, которая не влияет на оригинальность [3]. Представление кода учитывает только логику программы.

АСД – это структурное представление кода, очищенное от элементов синтаксиса конкретного языка программирования, где узлами являются операторы, к которым присоединяются их аргументы, а те в свою очередь тоже могут быть составными узлами. Преимуществом использования этого представления является большое количество парсеров для разных языков программирования, а также возможность

нахождения заимствований типов Т1 и Т2 с высокой точностью, т.к. порядок следования частей кода становится неважен. К недостаткам относятся сложность реализации и длительное время работы алгоритмов построения и анализа АСД.

3. Токенизированное представление [4].

Часто узлы АСД получаются из лексем, которые выделяются на этапе лексического анализа. Т.е. код преобразуется в последовательность лексических единиц с определенным значением, называемым токеном.

Токенизация осуществляется по следующему алгоритму:

– Каждому оператору языка программирования или группе операторов, имеющих схожее назначение, присваивается уникальный идентификатор. Значения идентификаторов назначаются заранее и для всех классов операторов.

– По полученным идентификаторам строится строка. В данной строке порядок токенов соответствует порядку следования их в исходном коде.

Такой подход позволяет игнорировать части кода, которые легко поддаются изменениям, а также выявлять заимствованные фрагменты кода, расположенные в разных местах программы.

Процесс токенизации зависит от конкретного языка программирования, т.е. поддержка нескольких языков возможна только при использовании нескольких лексических анализаторов. Также заимствования типа Т3 при использовании такого представления обнаруживаются с небольшой точностью.

4. Представление в виде метрик программы.

Представление основано на оценке различных метрик программы, в качестве метрик могут использоваться: количество вызовов функций, используемых переменных, циклов и т.д. Далее две программы сравниваются по соответствующим значениям метрик, если они близки или совпадают, то выявляется заимствование.

Данный метод не привязан к языку программирования, имеет высокую производительность реализующих алгоритмов и хорошую масштабируемость, но при этом в небольших программах, какими зачастую и являются учебные задания, метод выдает большое количество неверных данных для всех типов заимствования.

5. Представление в виде ориентированного графа.

Заимствования выявляются на основе графа зависимостей программы (ГЗП) – ориентированного графа, объединяющего информацию о потоке данных и потоке управления.

Вершины ГЗП представляют операции программы, а ребра представляют зависимости между ними. Ребра разделяют на зависимости по данным и по управлению. После составления ГЗП анализируемых программ, для них осуществляется поиск схожих подграфов. Подграфы признаются схожими, если они связны и имеют совпадающие множества ребер. Данный метод отличается тем, что граф хранит информацию и о семантике, и о структуре, способствуя высокой точности обнаружения заимствований программного кода типов Т1, Т2, Т3. Недостатком представления является плохая масштабируемость графа и длительное время его построения и анализа.

2. Выбор модели представления программ

Для учебной деятельности важны следующие аспекты нахождения заимствования программного кода:

- поддержка нескольких языков программирования;
- высокая точность для небольших по объему исходного кода программах;
- поддержка нахождения как можно большего количества типов заимствования.

Все три типа заимствований с высокой точностью обнаруживаются только при использовании метода построения ГЗП, но алгоритм анализа на основе данного метода требует значительного времени на выполнение, что не является приемлемым для массовой проверки большого количества работ в режиме реального времени при проведении различных контрольных и самостоятельных работ.

Для поиска заимствований типов Т1 и Т2 наилучшим является метод токенизации. Также возможно расширять количество поддерживаемых языков с помощью добавления для них словарей токенов. Для реализации может использоваться готовый генератор парсеров, что позволит упростить задачу анализа языка до написания грамматики в расширенной форме Бэкуса-Наура.

3. Проектирование метода

На рисунке представлена блок-схема предлагаемого алгоритма выявления плагиата в исходном коде программ.

Предлагаемый алгоритм состоит из следующих шагов:

1. Анализ языка программирования исходной программы и выбор словаря токенов. Словари строятся с учетом возможного сокрытия заимствования кода в виде замен эквивалентных операторов. Такие операторы будут записываться в словарь под одним идентификатором. Пример фрагмента словаря токенов для языка С++ представлен в таблице.

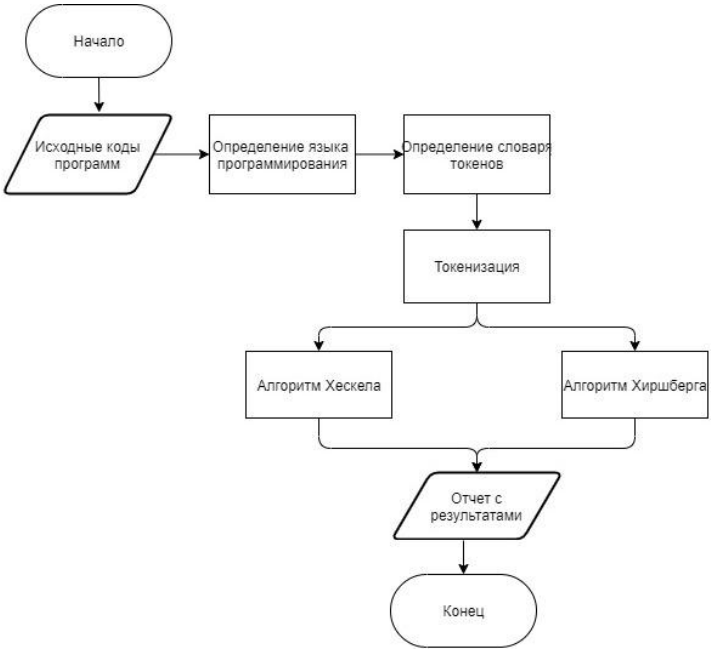

Рисунок. Метод выявления плагиата в исходном коде программ

Таблица

Пример фрагмента словаря токенов для языка C++

Идентификатор	Описание	Примеры
CYCLE	Циклы	for, do, while
CONDITION	Условные операторы	if, else
NUMBER	Числовые значения и константы	26, 4.2, -2.2e-8
TYPE	Типы переменных	bool, char, int, double
OPERATOR	Математические операторы и равенство	=, +, -, *, /, %
COMPARE	Операторы сравнения	==, >, <, >=, <=, !=
LOGIC	Логические операторы	&&, ||
BITS	Побитовые операторы	&, |, ^

2. Проведение лексического анализа (токенизация). Программный код преобразуется в строки токенов. Далее необходимо провести анализ

полученных строк для выявления в них общих фрагментов. Были проанализированы основные алгоритмы поиска плагиата в текстах и выбраны следующие два алгоритма: алгоритм Хескела и алгоритм Хиршберга, т.к. они имеют наименьшие показатели затрачиваемых времени и памяти при достаточно высокой точности результатов.

3. Алгоритм Хескела. На вход алгоритма даются две строки токенов, полученные на предыдущем этапе. Данные строки разбиваются на k-граммы (подстроки длины k). Для каждой пары k-грамм проверяется совпадение со строками, лежащими над ними. При совпадении проводится аналогичная проверка для этих строк, пока не будет найдено несовпадение. Также для строк, лежащих ниже k-грамм. Таким образом получаем множество непересекающихся подстрок.

Результатом работы алгоритма будет числовое значение – вероятностная мера Иверсена, аналог коэффициента Жаккара [5]. Данное значение можно считать приблизительной оценкой сходства двух программных кодов. Для удобства оценки значение получается в процентном виде.

$$K(A,B) = \frac{|A \cap B|}{|A \cup B|} * 100 \quad , \qquad (1)$$

где A, B – множества выявленных непересекающихся подстрок в сравниваемых программах.

4. Алгоритм Хиршберга [6]. Данный алгоритм является оптимизированной версией известного алгоритма Вагнера-Фишера, т.к. затраты памяти линейные, а не квадратичные. Он позволяет вычислить кратчайшее расстояние Левенштейна, которое отражает сходство между двумя строками [7].

Сходство двух строк измеряется как количество замен, вставки и удаления, необходимых для преобразования одной строки в другую. Расстояние Левенштейна увеличивается в зависимости от количества преобразований, необходимых для превращения одной строки в другую. Чем больше расстояние – тем меньше сходство двух строк.

Одна строка делится на две равные части, и при вычислении значений матрицы предыдущие столбцы удаляются. Таким образом заполняются обе части матрицы, а затем выбирается минимум сумм значений левой и правой части матрицы. Аналогично задача делится для подстрок, подразделение завершается, когда не останется ни одной подстроки длины больше 1. Получается список разбиений, на основе которого рассчитывается расстояние Левенштейна и вычислить значение, выражающее процент схожести двух программ.

$$K(A,B) = \left(1 - \frac{LD}{Max(A,B)}\right) * 100 \quad , \qquad (2)$$

где A, B – длины проверяемых строк, LD – расстояние Левенштейна.

5. Оценка заимствования. По завершении работы двух алгоритмов возвращаются результаты проверки по каждому из них. Чем больше определенный процент, тем больше вероятность того, что проверяемый программный код является плагиатом. Для окончательного определения необходимо высчитывать среднее значение результатов и установить пороговое значение, превышение которого будет означать, что выявлен плагиат.

Заключение

В данной работе рассмотрены основные типы заимствований программного кода. Проведен анализ способов представления исходного кода и алгоритмов выявления заимствований. На основе анализа предложен собственный алгоритм, основанный на токенизации и реализующий алгоритмы работы со строками Хескела и Хиршберга. Применение предложенного алгоритма позволяет определить процент схожести двух программ, который может быть использован для оценки присутствия плагиата.

Предложенный алгоритм учитывает поддержку нескольких языков программирования и требования к скорости проведения анализа для использования при проверке программ в рамках учебных курсов.

Список литературы

1. Roy C. K. A survey on software clone detection research / C. K. Roy, J. R. Cordy // Tech. Rep.2007-541, School of Computing. – Ontario, Canada, 2007. – С.14-18.

2. Зубов М.В. Подходы к статическому анализу открытого исходного кода / М.В. Зубов, А.Н. Пустыгин, Е.В. Старцев // Матер. 8-й Междунар. конф. “Linux Vacation” Eastern Europe. – Брест: Альтернатива, 2012. – С.36-40.

3. Лаздин А.В. / Метод построения графа функциональной программы для решения задач верификации и тестирования / А.В. Лаздин, О.Ф. Немолочнов // Научно-технический вестник информационных технологий, механики и оптики. – 2002. – №6. –С. 118-121.

4. Сорокин Д.С. / Увеличение стойкости метода токенизации к внесению изменений при обнаружении плагиата в исходном коде / Д.С.

Сорокин, А.Н. Шиков // Фундаментальные и прикладные исследования: проблемы и результаты. – 2014. – № 11. – С.132-136.

5. Iversen J. Über die Korrelationen zwischen den Pflanzenarten in einem grönlandischen Talgebiet / J. Iversen // Vegetation. – 1954. – №1. – С. 238-246.

6. Hirschberg D.S. A linear space algorithm for computing maximal common subsequences / D.S. Hirschberg // Communications of the ACM. – 1975. – №6. – С. 341-343.

7. Желудков А. В. Особенности алгоритмов нечёткого поиска / А. В. Желудков, Д. В. Макаров, П. В. Фадеев // Инженерный вестник. – 2014. –№12 – С. 501-510.