АНАЛИЗ ИСХОДНОГО КОДА НА ПРЕДМЕТ ЗАИМСТВОВАНИЙ В РАМКАХ ПРОЦЕССА АВТОМАТИЗИРОВАННОГО КОНТРОЛЯ ЗНАНИЙ

Я. О. Ярошевич

Белорусский государственный университет, г. Минск;
yaroshevich.yana@gmail.com;
науч. рук. – А. Н. Курбацкий, д-р техн. наук, проф.

В рамках данной работы построен и реализован комбинированный алгоритм идентификации заимствований. Он протестирован на работах студентов ФПМИ. Рассмотрены подходы к проверке кода на «чистоту».

Ключевые слова: идентификация заимствований; алгоритмы представления исходного кода; токенизация; коэффициент Жаккара; метод просеивания; метод отпечатков; комбинированный алгоритм.

ВВЕДЕНИЕ

В настоящее время актуальность автоматизированных тестирующих систем (АТС) возрастает в связи с развитием информационных технологий и дистанционного обучения. Проблема плагиата в образовании состоит не только в незаконном присвоении авторских трудов, но и в подрыве самой сути учебного процесса. Поэтому при создании АТС для проверки решений задач по программированию важным пунктом является выявление заимствований исходного кода.

В рамках АТС был разработан и реализован комбинированный алгоритм идентификации заимствований. Также были рассмотрены подходы для реализации автоматизированной проверки кода на «чистоту».

АЛГОРИТМ ИДЕНТИФИКАЦИИ ЗАИМСТВОВАНИЙ

Шаг 1. Предварительная обработка текста программы

В качестве языка программирования будем рассматривать C++. Для выполнения первого шага алгоритма производим удаление подключений библиотек, комментариев, лишних пробельных символов, неиспользуемых переменных, функций, классов, пустых и выводящих конструкций.

Шаг 2. Представление в виде элемента 25-мерного пространства

Представим программу как точку в 25-мерном пространстве, где i-ая координата является количественной характеристикой некоторого свойства программы. Если точки двух программ находятся на небольшом расстоянии друг от друга, то одна из них считается плагиатом другой.

Будем рассматривать следующие характеристики (метрики):

- Количество логических строк кода (LLOC) [1].
- Элементарные количественные метрики: число функций, классов.
- Метрики Холстеда: число уникальных операторов, уникальных операндов, общее число операндов, общее число операторов, размер словаря программы, размер программы, объем программы, уровень качества программирования, усилия для разработки/понимания программы, время на разработку/понимание программы, трудоемкость кодирования, уровень языка выражения, теоретическая длина программы, интеллектуальные усилия при разработке [2].
- Метрики Джилба: количество операторов цикла, условных операторов.
- Метрики графа управления: количество вершин, дуг.
- Цикломатическое число Мак-Кейба.
- Метрики Мейджела, Харрисона (SCOPE, SCORT).
- Метрика Чепина [3].

Токенизация

Основной идеей токенизированного представления является сохранение только существенных деталей исходного кода. Каждому оператору языка ставится в соответствие токен, описанный ранее для каждого класса операторов. Далее работаем с набором токенов, как со строкой.

Будем использовать следующие способы токенизации:

1. KIOV-токенизация: ключевые слова языка (K), идентификаторы (I), операторы языка (O), значения (литералы) (V).
2. Модификация 1: ключевые слова языка, циклы, встроенные типы переменных, одиночные операторы, двойные операторы, тройные операторы, значения (литералы), имена идентификаторов.
3. Модификация 2: круглые, квадратные, фигурные скобки, ключевые слова, строковые литералы, символьные литералы, числовые литералы, логические литералы, имена идентификаторов, операторы.

Шаги 3 - 5. Алгоритмы на токенизированном представлении кода

Для каждого типа токенизации применяем 3 алгоритма для рассматриваемой программы и каждой программы, сохраненной в базе данных:

1. Метод отпечатков [4] на основе метода просеивания [3] с минимальной длиной подстроки, равной 8, и длиной k-граммы, равной 6.
2. Вычисление коэффициентов Жаккара [5] с длинами *k*-грамм 3 и 4.
3. Метод нахождения наибольшей общей подстроки [3] с минимальной длиной значащей подстроки, равной 5.

Шаг 6. Принятие решения

Анализируем полученные 13 коэффициентов. Визуализируем участки, подозрительные на плагиат.

ПРОВЕРКА КОМБИНИРОВАННОГО АЛГОРИТМА

Для тестирования комбинированного алгоритма одиннадцати студентам ФПМИ БГУ было предложено решить несколько задач. При этом ставилось условие, что каждую задачу несколько студентов выполняют самостоятельно, а остальные студенты «списывают». В ходе тестирования все решения были загружены в АТС.

Полученные программным путем результаты подтверждают правильность разработки комбинированного алгоритма. Комбинированный алгоритм корректно работает на простых программах, что позволяет нам с уверенностью утверждать, что алгоритм может быть использован для определения заимствований в исходном коде любых программ.

Отметим, что принятие решения, является ли работа плагиатом, остается за преподавателем. А система находит и визуализирует результаты поиска заимствований, что облегчает процесс принятия решения.

ПРОВЕРКА ИСХОДНОГО КОДА НА «ЧИСТОТУ»

Сколько существует программистов, столько найдется и определений понятия «чистоты» кода. Известным фактом является то, что соотношение времени чтения и написания кода на достаточно масштабных проектах превышает 10:1. Поэтому важно, чтобы студенты, которые в будущем станут программистами, сразу учились писать «чистый» код.

Будем называть код «чистым», если он удовлетворяет некоторому стандарту оформления кода – набору правил и соглашений, используемых при написании исходного кода на языке программирования.

Стандарт оформления кода обычно строится так, чтобы за счёт определённого визуального оформления элементов программы повысить его информативность. Обычно, стиль программирования описывает:

- Способ расстановки фигурных скобок, ограничивающих логические блоки.
- Стиль отступов при оформлении логических блоков.
- Способы выбора названий и используемый регистр символов для имён переменных и других идентификаторов.
- Использование пробелов при оформлении логических и арифметических выражений.

- Стиль комментариев и использование документирующих комментариев.
- Ограничения размеров кода по вертикали и горизонтали.
- Отсутствие магических чисел и др.

Для реализации алгоритма форматирования кода исходный код представляется как синтаксическое дерево, затем узлы дерева последовательно выводятся с нужным количеством пробельных символов. Можно сравнивать полученный таким образом «чистый» код с исходной версией.

Для проверки кода на «чистоту» можно использовать рассмотренные метрики Холстеда, которые служат для оценки качества кода, и метрики стилистики и понятности программ (насыщенность программ комментариями). Также можно переиспользовать подготовительный этап: удаление неиспользуемых переменных и функций.

ЗАКЛЮЧЕНИЕ

В рамках данной работы был выполнен анализ различных алгоритмов представления исходных кодов программ и алгоритмов идентификации заимствований, а также был спроектирован и разработан собственный комбинированный алгоритм идентификации заимствований. С помощью разработанной автоматизированной системы контроля знаний было успешно проведено тестирование разработанного алгоритма поиска заимствований на работах студентов ФПМИ БГУ.

Библиографические ссылки

1. Software Testing Solutions for Productivity and Quality [Электронный ресурс] // URL: http://www.verifysoft.com/ (дата обращения: 06.03.2017).
2. Halstead complexity measures [Электронный ресурс] // URL:
3. https://en.wikipedia.org/wiki/Halstead_complexity_measures (дата обращения: 03.03.2017).
4. Программный код и его метрики [Электронный ресурс] // URL:https://habrahabr.ru/company/intel/blog/106082/ (дата обращения: 01.03.2017).
5. *N. Heintze*. Scalable document fingerprinting. In 1996 USENIX Workshop on Electronic Commerce, 1996.
6. *Steven Burrows, Seyed M. M. Tahaghoghi, and Justin Zobel*, Efficient and effective plagiarism detection for large code repositories // Proceedings of the Second Australian Undergraduate Students' Computing Conference (AUSCC04), pages 8–15, 2004.