ПОИСК ПЛАГИАТА В ПРОГРАММНОМ ОБЕСПЕЧЕНИИ

Сидоркина И. Г., Хукаленко К. С.
Факультет информатики и вычислительной техники, Поволжский государственный технологический университет
Кафедра информационно-вычислительных систем, Поволжский государственный технологический университет
Йошкар-Ола, Марий Эл
E-mail: igs592000@mail.ru, konstantin.khukalenko@gmail.com

Программирование характеризуется большим числом разнообразных правил, приемов, методов и средств его выполнения, применение которых зависит от квалификации, опыта и индивидуальных особенностей программистов. В статье анализируются особенности алгоритмов анализа плагиата программного кода, величины семантического шума в текстах программ на основе исследуемых методов. Приводится алгоритм с использованием комбинированного подхода некоторых текстовых и семантических алгоритмов. Показано, в какое представление переводится исходный код программ в большинстве современных алгоритмов, описаны классы современных алгоритмов поиска плагиата в исходных текстах программ. В результате предложен комбинированный алгоритм поиска плагиата для использования в учебной практике и для выявления плагиата среди программ-участников олимпиад. Полученный алгоритм объединяет в себе как плюсы текстовых алгоритмов, так и семантических, при этом основная вычислительная часть имеет очень хороший параллелизм, что сокращает время ее выполнения при наличии вычислительных мощностей.

Введение

Бурное развитие информационных технологий, значительно упростило для людей доступ к информации. Распространение сети Интернет привело к такому негативному явлению, как плагиат. Плагиат программных продуктов - вид нарушения прав автора, состоящий в незаконном использовании под своим именем, без указания ссылки на источник, чужого исходного кода, алгоритма или пользовательского интерфейса, а так же их частей. Плагиат в исходных кодах программ встречается как в коммерческой разработке программного обеспечения, так и в учебной практике. Проблема особенно актуальна в сфере высшего образования, которая заключается не только в присвоении чужих работ, но и подрыве самой сути образовательного процесса. Необходимость инструментов для выявления плагиата непосредственно связана и с защитой интеллектуальной собственности. Таким образом, задача выявления плагиата приобретает все большую актуальность в различных сферах человеческой деятельности, и как следствие необходимы методы и средства, позволяющие автоматизировать этот процесс.

Предлагается выделять три основные составляющие, относящиеся к программному обеспечению:

- Плагиат исходного кода — наиболее часто встречающаяся составляющая прикладного программного обеспечения (ППО). Заключается в прямом копировании исходного кода.
- Плагиат пользовательского интерфейса. Яркий пример представлен в мобильных устройствах. Из-за ограниченного форм-фактора данного класса устройств, наиболее удобные решения в области пользовательского интерфейса копируются производителями с незначительными изменениями.
- Плагиат алгоритма — в большинстве случаев является некоторой модификацией плагиата исходного кода.

Программы, написанные на олимпиадах по программированию, в первую очередь, отличаются слабой структурированностью кода. На подавляющем большинстве олимпиад в качестве решения принимается один файл, содержащий весь исходный код решения. Если проверять выявлять плагиат в условиях лишь одной олимпиады, то база аналогов относительно не большая. Но, не стоит исключать возможности сравнения посылок с результатами предыдущих соревнований. Большинство современных олимпиад позволяют участникам использовать различные языки программирования. При таком подходе, возможна ситуация плагиата алгоритма. В рамках удаленной индивидуальной олимпиады, в аудитории может находиться один участник, алгоритмически решающий задачи. Остальные же его «однокомандники» будут реализовывать данный алгоритм на различных языках, тем самым получая неоспоримое преимущество. Для решения некоторых задач, на олимпиадах не высокого уровня, применяются общеизвестные алгоритмы. Наличие функции «исключения алгоритма» могло бы значительно уменьшить число ложных обнаружений заимствования кода. Наличие высокой производительности у детектора плагиата олимпиадных задач является желательным условием. Проверка на наличие нарушений прав авторства требует окончательно-

го принятия решения человеком. Ситуация, при которой проверка на плагиат запускается после окончания приема решений олимпиадных задач, не требует выдающихся показателей производительности. Это объясняется тем, что организаторами может быть заранее предусмотрено время на проведение данного этапа приемки работ.

I. Специфика олимпиад по программированию

Интерес к олимпиадам по программированию и информатике среди студентов в России и во всем мире быстро растет. Успехи в олимпиадах наглядно демонстрируют степень подготовки программистов в странах и регионах, повышая авторитет и рейтинг соответствующих учебных заведений. В тех вузах, где развито "олимпиадное движение", возрастают интерес к компьютерным наукам, мотивация студентов к учебе, общий уровень программирования. На программистские специальности приходят профессионально ориентированные студенты. На студентов-олимпиадников с большим вниманием смотрят ведущие фирмы, занимающиеся разработкой программных продуктов.

Обычно считается, что сильный в программировании студент должен удачно выступать на олимпиадах. Это не так по следующим причинам:

1. Олимпиадник должен обладать выраженными спринтерскими качествами: быстро соображать, быстро и без ошибок набирать тексты, быстро тестировать и отлаживать программы. В обычном программировании не возникает потребности в такой срочности.
2. Необходимо в совершенстве владеть специальными разделами математики, которые изучаются как минимум недостаточно в стандартных учебных курсах. В первую очередь это арифметика целых чисел, комбинаторика (подсчет комбинаторных конфигураций, комбинаторика конечных множеств, перечислительные задачи комбинаторного анализа), поиск и сортировка, алгоритмы на графах (связность, кратчайшие пути, циклы, потоки в сетях и т. д.), перебор и методы его сокращения (динамическое программирование, метод ветвей и границ, метод "решета" и т. д.), вычислительная геометрия, элементы теории формальных грамматик и автоматов. Содружество программистов и математиков дает хорошие результаты.
3. По каждой теме необходимо решить достаточное количество задач, довести их до уровня работающих программ. Если, например, задача независимо от ее содержательной "упаковки" сводится после ряда преобразований к алгоритму нахождения кратчайшего пути в графе, то она считается решенной. Участник олимпиады, установив этот факт, всю остальную работу должен выполнять как автомат, она не должна требовать значительных усилий. Далее можно переключаться на следующую задачу.
4. Необходимы и систематические тренировки и опыт участия в олимпиадах. Невозможно быстро "натаскать" студентов, разбирая готовые решения.

Формально определить понятие плагиата при проведении олимпиады по программированию крайне непростая задача. Обычно под этим понимают случай, когда между исходными кодами двух программ есть существенная (на уровне языка программирования) общая часть. При этом производная программа получается из оригинальной несложными преобразованиями, цель которых – скрыть факт заимствования вставкой лишних операторов, изменением порядка следования в программе независимых операторов, разбиением одной функции на две, изменением имен переменных и т.п. Такое определение в большинстве случаев пригодно, т.к. серьезные изменения исходного кода, сделанные для сокрытия плагиата вручную, крайне трудоемки, если же для этого используются автоматические средства, то по виду исходного кода это легко определяется человеком.

Наиболее популярные системы автоматизированной проверки решений eJudge [Клопов И.Н.] и Contester [Чернов А.В.] являются web-сервисами. Автоматизированная система - детектор плагиата должен поддерживать интеграцию с подобными сервисами. Для этого необходимо помимо наличие удобного пользовательского интерфейса, и так же, гибкого интерфейса командной строки.

II. Классификация существующих алгоритмов поиска заимствований в программном коде

Существующие в данной области алгоритмы принято классифицировать следующим образом [Prechelt L., Malpohl G., Philippsen M.]:

- текстовые алгоритмы;
- структурные алгоритмы;
- семантические алгоритмы.

Текстовые алгоритмы представляют программу в виде строки над алфавитом, символы которого представляют оператор или группу операторов языка программирования. Причем аргументы операторов (которые сами могут быть операторами) игнорируются, что делает многие элементарные действия по сокрытию плагиата(например, изменения имен переменным) бесполезными. Символ такого алфавита традиционно называют токеном.

Текстовые алгоритмы [Huang X., Hardison R.C., Miller W.] включают в себя как наиболее эффективные современные алгоритмы поиска плагиата в исходных кодах программ, так и самые старые алгоритмы. Примером последних может служить подсчет некоторых характеристик программ (например, количество операторов ветвления в программе), а затем полученные векторы характеристик сравниваются между собой. Очевидна крайняя неэффективность такого подхода.

Практически во всех алгоритмах текстовой группы полагается, что если исходные коды программ являются похожими, то и на уровне строковых представлений у них присутствуют существенные общие части. Причем эти общие части не обязательно должны быть непрерывны. Из современных текстовых алгоритмов и методов выделим следующие:

- алгоритм на основе выравнивания строк - раздвигает символы строк так, чтобы выявить совпадающие участки;
- алгоритма основе строкового замощения [Wise M.J.]- эвристический алгоритм, выдающий наибольшее множество непересекающихся совпадающих подстрок, используются 2 эвристики:
  1. более длинные последовательные совпадения лучше, чем набор меньших и непоследовательных, даже если сумма длин последних больше;
  2. алгоритм игнорирует совпадения, длины которых меньше определенного порога
- метод просеивания ищет все общие подстроки фиксированного размера;
- метод отпечатков - сравнивает выборочные участки программ.

Структурные алгоритмы [Baxter I., Yahin A., Moura L., Anna M.S., BierL.], что следует из названия, используют саму структуру программы, обычно это или граф потока управления, или абстрактное синтаксическое дерево. Вследствие такого представления алгоритм сводит на нет многие возможные действия плагиатора. Но все алгоритмы этого класса являются крайне трудоемкими, поэтому не используются на практике.

Семантические алгоритмы [Moussiades L.M., Vakali A.]в чем-то похожи на структурные и текстовые, но в их основе лежат логические выводы, поиск в пространстве состояний. Например, в них может использоваться представление исходного кода программы в виде графа с вершинами двух типов. Одни строятся из последовательности операторов, которым назначается определенная семантика (например, математическое выражение, цикл), другие задают отношение, в котором состоят соседние с ней вершины (например, вхождение). Между двумя вершинами первого типа обычно стоит вершина второго.

III. Комбинированный подход

Предлагаемый подход объединяет в себе идеи некоторых текстовых и семантических алгоритмов. Ниже расписана последовательность действий алгоритма.

Семантическая часть

1. Преобразуем исходный код сравниваемых программ в поток токенов (токен–оператор или идентификатор языка программирования).
2. Разбиваем полученный поток токенов на слова, где под словом подразумевается простой оператор языка(в большинстве языков таким словом будет набор операторов между разделителями). В большинстве случаев исходный код, соответствующий одному слову занимает одну строчку.
3. Используя полученный набор слов, строим дерево, в узлах которого будут лишь те слова, которые соответствуют условным операторам, циклическим операторам или вызовам пользовательских функций, а листьями дерева будут остальные слова (а также пустое слово, например, для случая, когда в блоке выполнения условного оператора ничего нет).

При этом все циклические операторы (for, repeat, и т.д.) сводим к эквивалентному оператору while, все операторы условия (тернарный оператор '? :', switch, else, и т.д.) сводим к эквивалентному оператору if, оператор вызова пользовательской функции становится родительским для всех слов, соответствующих исходному коду данной функции.

Текстовая часть

1. Сравниваем соответствующие узлы следующим образом: каждый условный оператор первой программы с каждым условным оператором второй, аналогично с циклическими операторами и операторами вызова функций. Все прочие ветвления дерева внутри выбранного узла игнорируются, и сравнивается набор слов, входящий прямо или косвенно в выбранный для сравнения узел.

   Группу сравниваемы наборов слов обозначим блоком, коэффициент совпадения блоков будет рассчитываться по формуле

$$\sum_{n}^{i} \frac{k_i}{n},$$

   где k_i – коэффициент схожести i-го слова, а n–их количество.

Полученный коэффициент можно использовать в метрике для выделения блоков, потенциально скопированных (полностью или частично) из другой программы.

2. При сравнении двух наборов слов, сравнивается каждое слово одного набора с каждым словом из другого, получаем соответствующие коэффициенты схожести. Чтобы получить конечный коэффициент схожести для одного слова, выбираем максимальный коэффициент его схожести на любое из слов другого набора.
3. Чтобы сравнить два слова между собой и получить коэффициент схожести, показывающий, какая доля первого слова включена во второе, будем перебирать размер окна и 'двигать' его по первому слову, окно – подстрока первого слова.

Выполняются следующие шаги:

1. Размер окна k изменяем в следующем диапазоне: от длинны первого слова len до t (шумовой порог – совпадение подстрок данного размера и меньше считаются случайными, например, 2).
2. Двигаем окно по первому слову (шаг = 1 символ, соответственно количество шагов = len – k + 1)
3. Если подстрока, соответствующая положению окна первого слова, входим во второе, помечаем все соответствующие символы первой строки.
4. По достижении окном текущего размера k конца слова, могут возникнуть следующие ситуации:
   - Отмеченных символов нет (ни одна подстрока размера k первого слова не встречается во втором), коэффициент схожести при данном k равен 0.
   - В противном случае, текущий коэффициент для данного k в зависимости от количества помеченных символов линейно находится в интервале [min .. max], где min = k / len, max = 1 + 1/len – 1/k

– После перебора всех k выбираем наибольший коэффициент из полученных.

Выбор границ min и max обусловлен следующим:

– min – если было хоть одно совпадение подстроки длинны k, то логичным будет вывод о том, что хотя бы k /len часть первой строки совпадает с частью второй
– max – при уменьшении размера окна, увеличивается шумовой эффект, уменьшаем полученный коэффициент схожести

IV. Заключение

Предложенный алгоритм объединяет в себе как плюсы текстовых алгоритмов, так и семантических. При этом основная вычислительная часть имеет очень хороший параллелизм, что сокращает время ее выполнения при наличии высокопроизводительных вычислительных мощностей. Стоит отметить, что эмпирических данных о работе данного алгоритма еще недостаточно, но для приемлемой скорости выполнения (до 2 сек, при сравнении двух программ до 1000строк исходного кода) достаточно и обычного ПК, при выполнении основной части программы на графическом процессоре. Программное обеспечение обработки результатов тестирования должно быть интегрировано с программной оболочкой тестирования.

V. Список литературы

1. Клопов, И. Н. Contester // CONTESTER.RU: официальный сайт 2012. [Электронный ресурс] URL: http://www.contester.ru (дата обращения: 20.11.2012).
2. Чернов, А. В. Система тестирования ejudge // Сборник материалов конференции свободное программное обеспечение в высшей школе. 2007. [Электронный ресурс] URL: `http://heap.altlinux.org/pereslavl2007/chernov/abstract.html` (дата обращения: 21.11.2012)
3. Wise, M. J. String similarity via greedy string tiling and running Karp-Rabin matching. // Dept. of CS, University of Sydney. December 1993.
4. Baxter, I. BierL. Clone Detection Using Abstract Syntax Trees. / I. Baxter, A. Yahin, L. Moura, M. S. Anna// Proceedings of ICSM. IEEE. 1998.
5. Prechelt, L. JPlag: Finding plagiarisms among a set of programs. / L. Prechelt, G. Malpohl, M. Philippsen // Technical Report No. 1/00, Universityof Karlsruhe, Department of Informatics. March 2000.
6. Moussiades, L. M., Vakali, A. PDetect: A Clustering Approach for Detecting Plagiarism in Source Code Datasets. / L. M.Moussiades, A. Vakali // The Computer Journal Advance Access. June 24, 2005
7. Manber, U. Finding similar files in a large filesystem. // Proceedings of the USENIX Winter 1994 Technical Conference. San Francisco. 1994. P. 1–10.
8. Huang, X. A space-efficient algorithm for local similarities. / X. Huang, R. C. Hardison, W. Miller // Computer Applications in the Biosciences 6. 1990. P. 373–381.