Хаустов Павел Александрович, магистрант кафедры вычислительной техники Института кибернетики ТПУ.
E-mail: eXceibot@sibmail.com
Область научных интересов: разработка и исследование алгоритмов сортировки и поиска.

Кацман Юлий Янович, канд. техн. наук, доцент кафедры вычислительной техники Института кибернетики ТПУ.
E-mail: katsman@tpu.ru
Область научных интересов: компьютерное моделирование сложных физических процессов и статистическая обработка результатов исследований.

УДК 519.172

СИСТЕМА ОБНАРУЖЕНИЯ ИСПОЛЬЗОВАНИЯ ЧУЖОГО ПРОГРАММНОГО КОДА «NoCrib»

П.А. Хаустов, Ю.Я. Кацман

Томский политехнический университет
E-mail: exceibot@sibmail.com; katsman@tpu.ru

Предложен алгоритм с использованием trie-дерева, позволяющий реализовать сравнение исходных кодов программ на языках программирования высокого уровня. Реализована система проверки программ «NoCrib», организована база данных для хранения исходных кодов под управлением MySQL. Система «NoCrib» реализует архитектуру «клиент-сервер». Для удобства работы пользователя разработан дружественный интерфейс.

Ключевые слова:
Система, плагиат, алгоритм, trie-дерево, NoCrib, лабораторные работы, сравнение.

Key words:
System, crib, algorithm, trie-tree, NoCrib, laboratory works, comparison.

Введение

В настоящее время при широком внедрении и общедоступности информационных технологий случаи плагиата не являются такой уж редкостью: студенты и молодые ученые используют чужие статьи и материалы в научных целях без ссылки на их авторов.

В конце 2010 г. на проблему плагиата обратило внимание руководство Московского государственного университета. Спустя несколько месяцев на сайте Томского политехнического университета появилась новость о том, что ТПУ поддерживает эту идею. Предполагается, что университет закупит специальное программное обеспечение, которое позволит отслеживать случаи плагиата при выполнении курсовых, выпускных и любых других индивидуальных работ, выполняемых студентами.

При написании квалификационной работы, так или иначе связанной с программированием, можно совершить особый вид плагиата, который заключается в неправомерном использовании чужого исходного кода программы. В таком случае можно также утверждать, что квалификационная работа частично или полностью выполнена другим лицом.

Проблема в том, что обнаружить плагиат в этом случае, более трудная задача, нежели обнаружить плагиат в текстовом материале. При сравнении исходных кодов программ необходимо анализировать суть исходного кода, ведь перестановка строк и замена имен переменных в коде программы приведут к значительным изменениям во внешнем виде кодов, но не приведут к изменениям в структуре кода и функциональном устройстве программы.

Постановка задачи

Целью проекта «NoCrib» является разработка и реализация системы с использованием алгоритма, позволяющего сравнить два исходных кода написанных на одном и том же языке программирования и релевантная оценка степени схожести этих кодов.

Многие алгоритмы нечеткого сравнения строк ориентированы на обычный текст и не учитывают специфические особенности сравнения исходных кодов программ. В связи с этим требуется алгоритм, удовлетворяющий следующим требованиям:

- алгоритм должен распознавать хотя бы основные ключевые слова языка;
- переменные не должны считаться различными, если они отличаются только именем;
- при обмене местами двух функций код не должен распознаваться как совершенно другой;
- аналогичный принцип применим для строк внутри функций: существуют строки, перестановка которых не приводит к изменению функциональности программы;
- при анализе все комментарии в коде должны игнорироваться, аналогично игнорируются пробельные символы и переводы строк.

Система «NoCrib» реализует нечеткое сравнение двух исходных кодов и выводит оценку (процент схожести этих кодов), согласно вышеприведенным требованиям. При анализе исходного кода программы алгоритм помимо своей функциональности должен отличаться незначительным временем выполнения и потреблять допустимое количество вычислительных ресурсов (памяти).

Анализ и разработка алгоритма сравнения

При сравнении кодов программ использовались алгоритмы, основанные на построении различных типов деревьев [1, 2]. При выборе и анализе различных алгоритмов особое внимание было уделено построению trie-деревьев исходного кода [3]. Такие деревья используются в программировании для хранения словаря, в вершинах этих деревьев находятся символы используемого алфавита. В данном случае хранить код как набор слов нерационально, так как многие слова являются лишь именами переменных или функций, и, в случае замены имен функций и переменных, полученный исходный код будет рассматриваться как код абсолютно непохожий на изначальный.

Очевидно, что рассматривать символ используемого алфавита как атомарную единицу кода нецелесообразно. Поэтому был предложен алгоритм, в котором в качестве атомарной единицы используются лексемы языка программирования. Если разобрать исходный код на лексемы и каждой из них присвоить индивидуальный идентификатор, то имена переменных, функций, значения математических констант перестанут играть роль в анализе схожести двух исходных кодов. На первый план выйдет именно структура программы: объявленные переменные, функции и их сигнатура, операции и операторы.

Рассмотрим подробнее, как хранится исходный код, разобранный на лексемы, в структуре данных trie-дерево. Каждый оператор в исходном коде заканчивается символом «;». Он может находиться в теле какой-либо функции или какого-либо цикла, который, в свою очередь, находится в теле какой-то функции. Следовательно, если рассматривать каждую пару операторных скобок как отдельный уровень вложенности, то можно построить дерево следующим образом: вершина с последней лексемой каждого уровня вложенности после добавления в trie-дерево становится корнем для поддеревьев всех операторов и последующих вложенных операторных скобок.

Рассмотрим для примера следующий исходный код на языке C++:

```
int f(int x)
{
    return x & 1 + 1;
}

int main()
{
    int x = 100, k = 0;
    while (x)
    {
```

```
            x -= f(x);
            ++k;
        }
        return 0;
    }
```

Упрощенное trie-дерево лексем (разработанный алгоритм), построенное по этому коду, представлено на рис. 1:

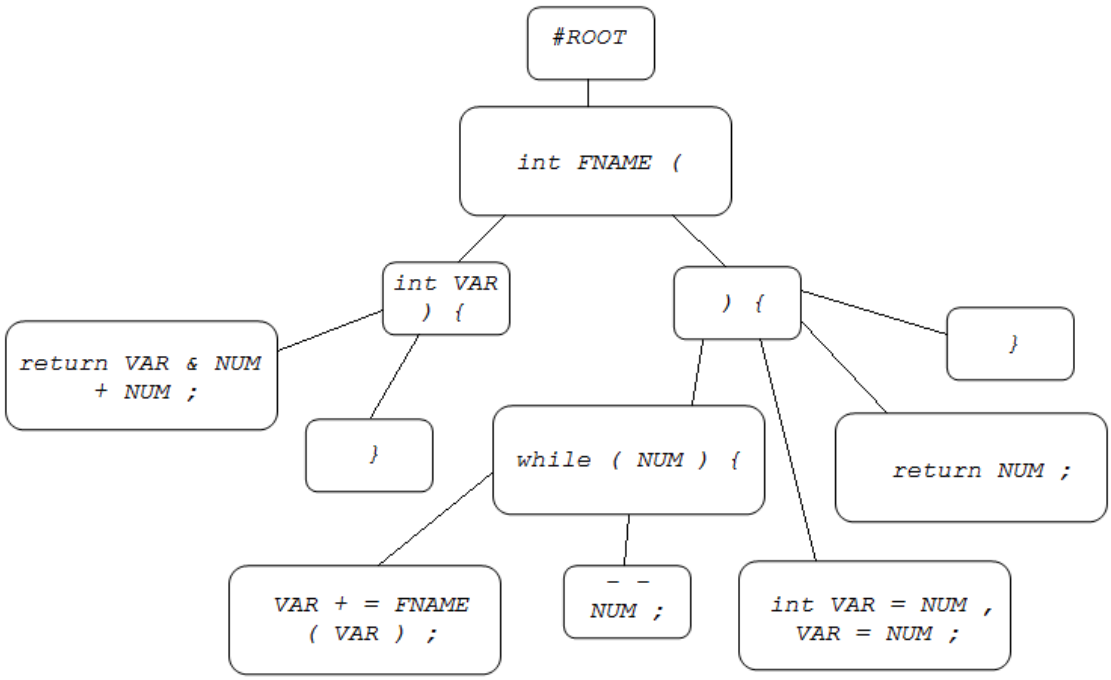

Рис. 1. Упрощенное дерево лексем

На рис. 1 некоторые вершины (блоки) содержат несколько лексем. В действительности каждая из этих лексем соответствует единственной вершине. Здесь приняты обозначения: VAR – имя переменной, NUM – арифметическая константа, FNAME – имя функции.

В программной реализации в узлах дерева хранятся идентификаторы лексем, что позволяет существенно уменьшить объем потребляемой памяти и увеличить быстродействие программы.

После того как дерево лексем для одного из кодов построено, оно используется для сравнения данного кода с другим. Если второй исходный код также разбить на лексемы и затем на уровни вложенности, то для каждого оператора можно обнаружить есть ли его аналог в дереве лексем первого исходного кода. Причем, проверяться будут все уровни вложенности на пути к этому оператору.

Стоит отметить существенную особенность предложенного алгоритма: при сравнения второго кода с первым не требуется создавать trie-дерево лексем для второго кода, так как в любой момент времени требуется хранить в памяти не более одной ветки оператора второго кода (то есть самого оператора и всех предшествующих ему уровней вложенности представленных в виде последовательности лексем). Для ветки каждого оператора второго кода можно идти по дереву лексем первого кода обходом в глубину от корня и смотреть есть ли точно такая же ветка в этом дереве. Очевидно, что если вся ветка отсутствует в дереве, то при обходе из корня возникнет ситуация, когда у текущей вершины среди детей нет вершины с нужным идентификатором лексемы.

Именно эта особенность предложенного алгоритма и легла в основу оценки схожести кодов программ. Обозначим количество вершин в самой ветке (эталонный код) за Y. Количество вершин, посещенных до этого момента в сравниваемой программе (второй код),

обозначим за X. Очевидно, что если в дереве присутствует искомая ветка, то X = Y, в остальных случаях X < Y. Следовательно, чем меньше число X, тем меньше совпадение данной ветки лексем второго кода с веткой дерева лексем первого кода. Для того чтобы с ростом X росла существенность ошибки, в оценке используются квадраты значений X и Y.

Для оценки общей схожести исходных кодов используется значение x = C / Z, где C – сумма X^2 для всех веток, Z – сумма Y^2 для всех веток. Функция оценки схожести кодов имеет вид:

$$F(x) = 100 \cdot \sqrt{x - 3\sqrt{(1-x) \cdot x}}$$

и принимает значения от 0 (абсолютная различимость) до 100 (абсолютная идентичность).

Программная реализация

Данный алгоритм реализован на языке C++ в виде консольного приложения и используется в системе «NoCrib». На вход приложению поступает два исходных кода для сравнения, а на выход подается величина процента схожести с 6 знаками после десятичной точки.

Система реализована на языке программирования PHP и представляет собой реализацию архитектуры «Клиент – Сервер». Пользователю предоставляется удобный web-интерфейс (см. рис. 2). Для хранения данных об учетных записях и базы исходных кодов была организована база данных под управлением СУБД MySQL.

Основным языком web-сайта выбран английский язык. В дальнейшем планируется осуществить выбор языка для работы в системе.

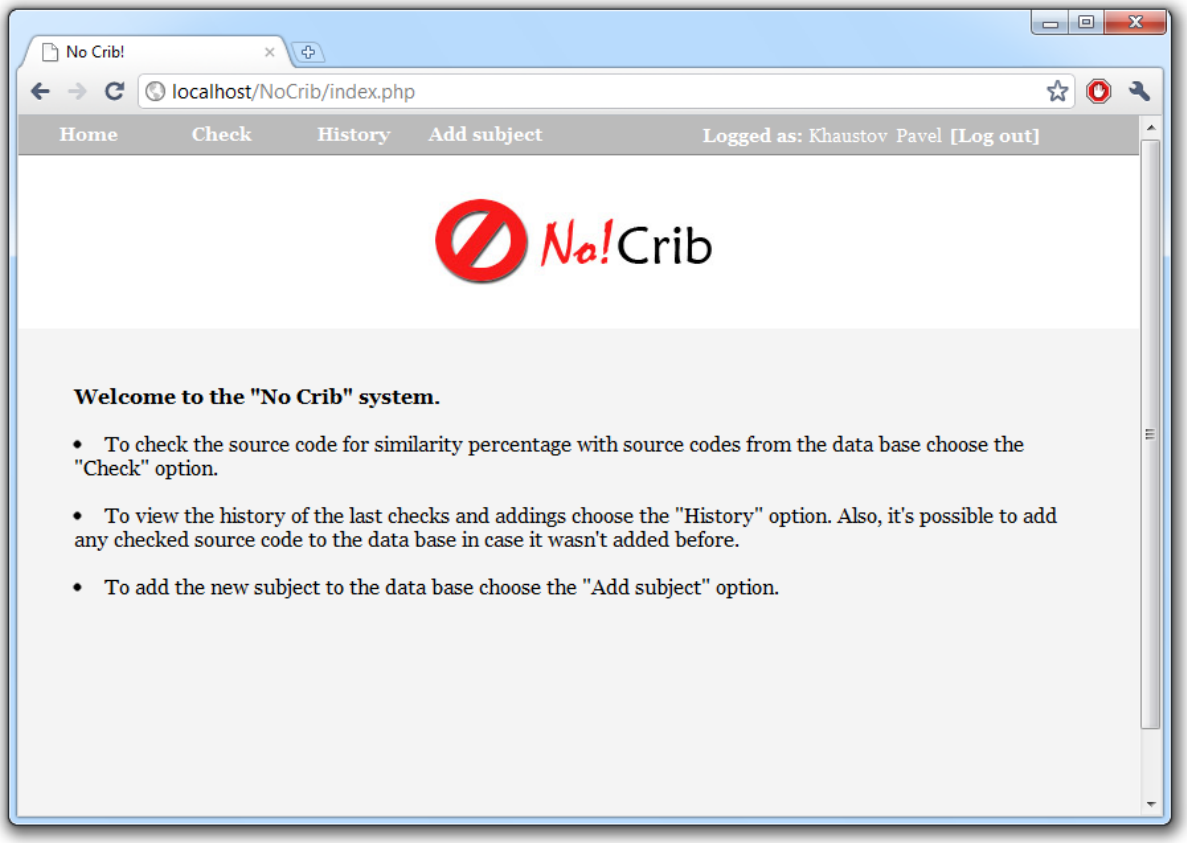

Рис 2. Общий вид пользовательского интерфейса

На главной странице пользователю сразу же предлагаются краткие инструкции по работе с системой. Для работы в системе необходимо авторизироваться. После авторизации на месте формы для входа в систему появляется информация о пользователе и ссылка для окончания сессии работы с системой.

Авторизированный пользователь может отправить на проверку код для сравнения с другими кодами (программами) имеющимися в базе данных. Для каждого исходного кода в соответствующем разделе отображается наиболее похожний на него код, и вычисляется процент схожести этих двух кодов.

Каждую из отправок можно добавить в базу данных как код, с которым в будущем можно будет производить сравнение. Если код уже получил схожесть большую, чем 99 % или был ранее добавлен в базу, то добавление такого кода невозможно.

Для каждого кода необходимо задать его тематику, то есть обозначить задачу, для решения которой он был написан. Тематика должна выбираться из базы данных. Добавление новой тематики в базу также возлгатается на пользователя системы, облажающего соответствующими правами. В целях увеличения быстродействия системы коды с различными тематиками не сравниваются.

Заключение

В будущем планируется реализация возможности распараллеливания процесса проверки на многопроцессорной системе (кластере). При такой реализации можно будет лишь единажды строить дерево для одного исходного кода, а затем для всех кодов, с которыми необходимо произвести сверку, проводить ее на отдельном процессоре. При таком подходе каждый поток будет обращаться к общему ресурсу – дереву с запросами на нахождение той или иной ветки в нем. Стоит отметить, что после построения дерево ни разу не изменится, то есть после построения дерево станет структурой данных доступной исключительно для чтения.

Система «NoCrib» может найти широкое применение для обнаружения нарушения самостоятельности выполнения лабораторных, индивидуальных и квалификационных работ, связанных с написанием программ.

СПИСОК ЛИТЕРАТУРЫ

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. – М.: МЦНМО, 2001. – 1293 с.
2. Кнут Д. Исскуство программитования для ЭВМ. Т. 1. Основные алгоритмы. – М.: Мир, 1976. – 729 с.
3. Бентли Д., Седжвик Р. Деревья тернарного поиска. – М.: Аддисон-Весли, 1998. – 480 с.

Поступила 15.09.2011 г.