УДК 004.912

Е. В. Малышев, В. В. Смелов
Белорусский государственный технологический университет

АЛГОРИТМ РАСПОЗНАВАНИЯ ПЛАГИАТОВ КОДОВ ПРОГРАММ

В данной статье рассмотрен алгоритм распознавания плагиатов кодов программ. Предлагаемый алгоритм позволяет оценить схожесть программных текстов и построить соответствующие кластеры. Описан подход к разработке программного обеспечения, реализующего данный алгоритм, сформулированы задачи, решаемые таким программным обеспечением. Основой алгоритма распознавания плагиатов является алгоритм жадного строкового замощения, который дает хорошие результаты сравнения текстов. Кластеризация осуществляется над графом, в котором каждой вершине сопоставляется исследуемый код программы, а кластер представляет собой подмножество вершин, соответствующее подмножеству схожих программ. Граф является взвешенным: каждому ребру ставится в соответствие число от 0 до 1, выражающее степень схожести двух программ. Областью применения программного обеспечения, разработанного на основе подходов, описанных в статье, являются системы автоматизированного тестирования, применяемые при проведении соревнований по спортивному программированию.

Ключевые слова: хэш, плагиат, токенизация, копирайт, кластеризация, теория алгоритмов.

Ye. V. Malyshev, V. V. Smelov
Belarusian State Technological University

PLAGIARISM DETECTING ALGORITHMS FOR SOFTWARE CODE

In this article the algorithm for recognizing plagiarism of program codes is considered. The proposed algorithm makes it possible to evaluate the similarity of program texts and construct the corresponding clusters. The approach to the development of software that implements this algorithm is described and problems which such software is able to solve are formulated. The basis of the algorithm for recognizing plagiarism is the algorithm of greedy string tiling, which gives good results for comparing texts. The process of clustering is performed over a graph in which each vertex is associated with the studied program code, and a cluster is a subset of vertices corresponding to a subset of similar programs. The graph is weighted: each edge is associated with number from 0 to 1, expressing the degree of similarity of two programs. The application of the software based on the article's approaches is automated test systems for sport programming competitions.

Key words: hash, plagiarism, tokenization, copyright, clustering, theory of algorithms.

Введение. Плагиат в программировании − умышленное присвоение авторства чужого фрагмента кода программы. Под кодом программы подразумевается текст, написанный на каком либо языке программирования или языке разметки. Для определения плагиата в исходных кодах программ используются алгоритмы, основывающиеся на различных факторах [1], таких как количество определенных операторов, стилистика написания кода, структура программы и др. В связи со спецификой учебного процесса [2] появляется острая необходимость разработки программной системы, позволяющей легко обнаружить факт заимствования или упростить исследование большого набора данных для обнаружения плагиата.

На данный момент не существует хорошего программного обеспечения для распознавания плагиата и поэтому данная тема все еще остается актуальной. Подход с кластеризацией, на основе схожести программ, поможет значительно упростить этот процесс поиска и позволит быстрее [3] распознавать факт заимствования. Так как каждый кластер в большинстве случаев представляет собой небольшую подгруппу программ, то это позволяет сфокусироваться на небольшом количестве данных и гораздо точнее и быстрее их проанализировать.

Основная часть. Большое количество программ неудобно для исследования на факт заимствования [3]. Для упрощения задачи будет целесообразным разбить весь набор исходных кодов на небольшие подгруппы. Каждая подгруппа должна содержать исходные коды, которые заимствовали части друг друга. Процесс разбиения группы исходных кодов на подгруппы называется кластеризацией. А сама подгруппа – кластером.

Процесс кластеризации производится над графом, вершинами которого являются программы. Каждому ребру задан вес от 0 до 1. Вес выражает степень схожести двух программ. Процесс построения ребер основывается на попарном сравнении программ. Ребро включается

в граф, если является значимым. Значимыми называются ребра, вес которых превышает заданный нижний порог. Для определения веса ребра применяется алгоритм жадного строкового замощения [4]. Используя токенизированное представление [5] каждой программы, можно улучшить точность веса ребра. Для улучшения временной оценки применяется алгоритм *Карпа – Рабина* [6].

Для формирования кластеров постепенно повышается нижний весовой порог, тем самым уменьшается количество значимых ребер. При повторении процесса увеличивается количество связанных компонент и уменьшается размер каждой. По достижению требуемого размера связанных компонент процесс останавливается. В дальнейшем над каждой группой вершин производятся дополнительные исследования.

Подходы оценки близости. В исходном коде программы сохраняются характеристики [2], свойственные автору (наименования, переменные, стиль исходного кода, комментарии и другие особенности). На основе особенностей можно выделить различные подходы для определения похожести двух исходных кодов (расстояние Левенштейна, атрибутный, структурированный, трансляторный, поведенческий, комбинированный). Предлагается использовать подход, основанный на алгоритме жадного строкового замощения [4], позволяющий просто определять степень похожести двух программ.

Модели представления исходных кодов программ. Токенизация. Для приведения исходного кода к общему виду, производится преобразование текста к набору токенов (токен − значащая часть исходного кода, как операторы, строки и т. д.). Процесс можно упростить, применив пакет Java Compiler Compiler (JavaCC).

Для работы с пакетом JavaCC создается файл с грамматическим описанием языка, а затем запускается на выполнение встроенный плагин [5]. В результате будет получен пакет классов, позволяющий работать с входным потоком по заданным правилам и преобразовывать его в набор токенов. Принципы описания грамматики в JavaCC во многом сходны с EBNF (расширенной формой Бакуса Наура), которая используется для описания формальных контекстно-свободных грамматик.

Ниже представлен JavaCC-код программы, удаляющей комментарии.

Алгоритм жадного строкового замощения и его улучшение. Для определения веса ребра между вершинами графа воспользуемся алгоритмом жадного строкового замощения (The Greedy String Tiling). Для этого сначала преобразуем программы в наборы токенов. В результате алгоритм выдаст набор общих непересекающихся подстрок. Подстроки, входящие в этот набор, называют тайлами (*tiles*). Обозначим исходные коды двух программ как P и T, введем следующие понятия:

• *MinMatchLen* − минимальная длина наибольшего общего префикса P_p и T_t;

• *MaxMatch* − длина самого большого из пока найденных префиксов;

• *Matches* − множество, содержащее кандидатов на попадание в набор тайлов;

• *Tiles* − набор тайлов.

```
SKIP : {
  // Однострочный комментарий
  < "//" (~["\r", "\n"])* >
  // Начало многострочного
  // комментария
  | < "/*" > : ML_COMMENT_STATE
}
// Игнорируем все, пока не найдем
// конец комментария
<ML_COMMENT_STATE> SKIP : {
  < "*/" > : DEFAULT
  | < ~[] >
}
```

Процесс поиска состоит из двух повторяющихся фаз. В первой фазе осуществляется поиск наибольших общих подстрок P и T, состоящих только из свободных элементов (изначально все элементы свободны). Для этого используются три вложенных цикла: первый из которых пробегает по всем возможным значениям P_p, второй цикл по всем T_t, а третий цикл находит наибольший общий префикс (НОП) P_p и T_t. В итоге будет найден набор всех префиксов с максимальной длиной, состоящих из свободных элементов. Вторая фаза проходит по списку найденных НОП. Если текущий элемент списка − подстрока из свободных элементов, то помещаем ее в выходной набор *tiles* и помечаем все ее элементы. Таким образом она не учитывается в дальнейшем. Если найденная максимальная длина все еще больше *MinMatchLen*, то переходим к первой фазе. Ниже приведем псевдокод данного алгоритма.

```
Greedy-String-Tiling (P, T) {
  tiles = {};
  do {
    maxmatch = MinMatchLen;
    For unmarked tokens Pp in P, Tt in T {
      j = 0;
      while (Pp + j == Tt + j) &&
        unmarked(Pp + j) &&
        unmarked(Tt + j) j ++;
      if (j == maxmatch) {
        matches ∪= match(p,t,j);
```

```
        } else if (j > maxmatch) {
          matches = {match(p,t,j)};
          maxmatch = j;
        }
      }
      For match(p,t,maxmatch)∈matches {
        if (not occluded) {
          for j = 0...(maxmatch−1) {
            mark(Pp + j), mark(Tt + j);
          }
          tiles ∪= matches(a,b, maxmatch);
        }
      }
    } while (maxmatch > MinMatchLen);
    return tiles;
  }
```

Константа *MinMatchLen* вводится, чтобы не учитывать слишком маленькие префиксы, которые негативно влияют на определение веса ребра.

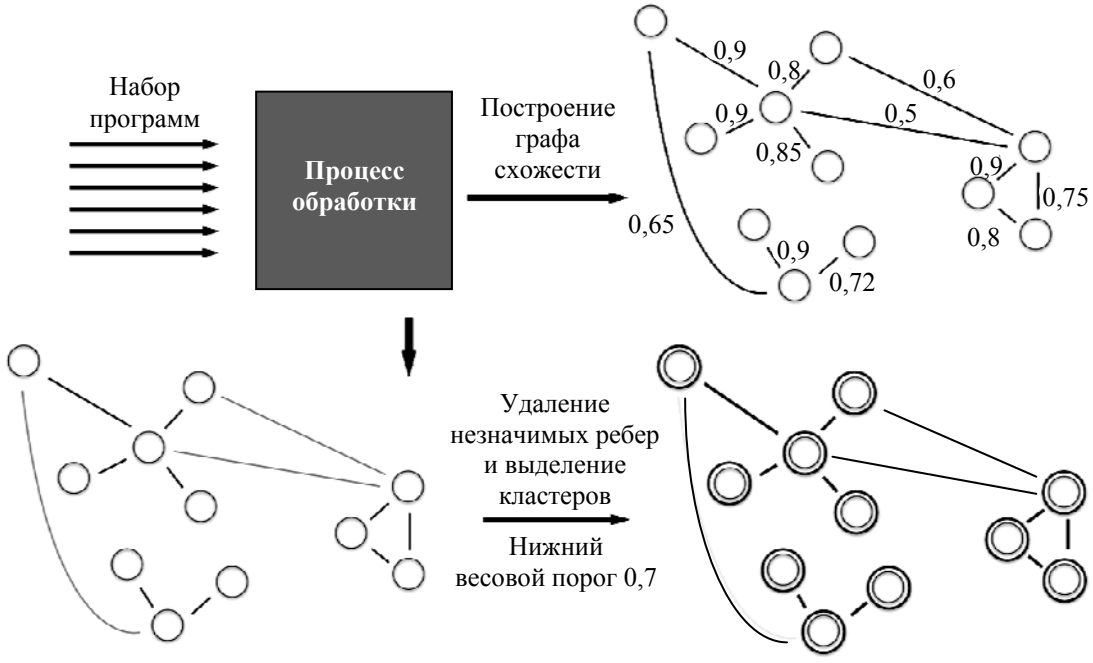
Проверку на то, что вся подстрока свободна (фаза 2) можно производить за время $O(1)$, так как достаточно проверить конечные элементы, и тогда временная сложность алгоритма $O(n^3)$. Оценку можно улучшить, используя алгоритм *Карпа – Рабина* [6] поиска подстроки в строке, применив подход по использованию хэшей. Будем вычислять значение хэш-функций для всех подстрок длины L в P и T, где L − параметр, больший или равный *MinMatchLen*. Это можно сделать за $O(|P| + |T|)$, используя полиномиальную хэш-функцию. Каждое хэш значение от подстроки P сравнивается с каждым значением от подстроки T. Если два значения одинаковы, то возможно равенство соответствующих подстрок, которое можно проверить посимвольно. Если происходит совпадение, то предпринимается попытка продления на длину большую, чем L, то есть ищется НОП соответствующих подстрок. Далее алгоритм выполняет оригинальные шаги. Асимптотика худшего случая после этой модификации составляет $O(n^3)$, но на практике она значительно ниже $O(n^2)$. После того как найден набор тайлов, вес ребра можно определить следующим образом:

$$\mathrm{sim}(\mathrm{P},\ \mathrm{T}) = \frac{2\,|\,\mathrm{tiles}\,|}{|\mathrm{P}|+|\,\mathrm{T}\,|},$$

где $|tiles|$ − длина всех найденных непересекающихся общих подстрок, каждая из которых длиной не меньше *MinMatchLen*, а $|P|$ и $|T|$ − длина программ P и T соответственно.

Кластеризация на основе схожести программ. Подход по выделению кластеров состоит из следующих шагов (рисунок):

1. На вход получаем набор программ.
2. Представляем каждую программу в виде набора токенов, получившихся на основе исходного кода.
3. Производим попарное сравнение, чтобы определить вес ребер и оставить только значимые из них.
4. Постепенно поднимаем нижний весовой порог, пока размер каждой компоненты связанности не будет удовлетворять нашим некоторым критериям.
5. Выделяем компоненты связанности, каждую из которых считаем отдельным кластером для дальнейшего исследования.



Процесс кластеризации кодов программ на основе степени похожести

Заключение. В разработанном программном обеспечении для поиска плагиата в наборе программ применяется алгоритм жадного строкового замощения. В качестве его улучшения используется подход *Карпа – Рабина*. Для выделения групп применяется алгоритм кластеризации на основе схожести программ. Практической полезностью разработанного программного обеспечения является автоматизация процесса проверки набора кодов программ. Разработанное программное обеспечение можно использовать в учреждениях образования, автоматических системах тестирования для проверки различных данных.

Литература

1. Chen X., Francia B., Li M., McKinnon B., Seker A. Shared Information and Program Plagiarism Detection // IEEE Trans. Information Theory. July 2004. P. 1545–1550.

2. Faidhi. J. A. W., Robinson. S. K. An Empirical Approach for Detecting Program Similarity within a University Programming Environment. Computers and Education. 1987. No. 11 (1). P. 1119.

3. Manber U. Finding similar files in a large file system // Proceedings of the USENIX Winter 1994 Technical Conference, San Francisco, CA, USA, 1994. P. 1–10.

4. Khurram Z., Tabassam N., Sami D., Ali J. Efficient Source Code Plagiarism Identification Based on GST // IJCSNS. December 2010. Vol. 10. No. 12. P. 204–210.

5. Copeland T. Generating Parsers with JavaCC. Alexandria: Centennial Books, 2007.

6. Cormen H., Charles E., Ronald L., Stein C. The Rabin–Karp algorithm. Introduction to Algorithms. Cambridge, Massachusetts: Mit Press, 1990.

References

1. Chen X., Francia B., Li M., McKinnon B., Seker A. Shared Information and Program Plagiarism Detection. *IEEE Trans. Information Theory*, July 2004, pp. 1545–1550.

2. Faidhi. J. A. W., Robinson. S. K. An Empirical Approach for Detecting Program Similarity within a University Programming Environment. *Computers and Education*, 1987, no. 11 (1), pp. 1119.

3. Manber U. Finding similar files in a large file system. *Proceedings of the USENIX Winter 1994 Technical Conference*, San Francisco, CA, USA, 1994, pp. 1–10.

4. Khurram Z., Tabassam N., Sami D., Ali J. Efficient Source Code Plagiarism Identification Based on GST. *IJCSNS*. December 2010, vol. 10, no. 12, pp. 204–210.

5. Copeland T. Generating Parsers with JavaCC. Alexandria, Centennial Books, 2007.

6. Cormen H., Charles E., Ronald L., Stein C. The Rabin–Karp algorithm. Introduction to Algorithms. Cambridge, Massachusetts, Mit Press, 1990.

Информация об авторах

Малышев Егор Вадимович – магистрант. Белорусский государственный технологический университет (220006, г. Минск, ул. Свердлова, 13а, Республика Беларусь). E-mail: wsemirz@gmail.com

Смелов Владимир Владиславович – кандидат технических наук, доцент кафедры информационных систем и технологий. Белорусский государственный технологический университет (220006, г. Минск, ул. Свердлова, 13а, Республика Беларусь). E-mail: smw60@mail.ru

Information about the authors

Malyshev Yegor Vadimovich – Master's degree student. Belarusian State Technological University (13a, Sverdlova str., 220006, Minsk, Republic of Belarus). E-mail: wsemirz@gmail.com

Smelov Vladimir Vladislavovich − PhD (Engineering), Assistant Professor, the Department of Information Systems and Technologies. Belarusian State Technological University (13a, Sverdlova str., 220006, Minsk, Republic of Belarus). E-mail: smw60@mail.ru

Поступила 18.12.2017