УДК 681.3.016

АМОНС А.А.,
ЗАЙЦЕВ С.Ю.,
КИРИЧЕК А.А.

ВЫЯВЛЕНИЕ ПЛАГИАТА В ПРОГРАММНОМ КОДЕ C#

В статье рассматривается вопрос о том, что такое плагиат. С начала описываются модели представления программ, а так же рассматриваются алгоритмы определения плагиата в программном коде C#. По каждому алгоритму проведен анализ и сравнение для выяснения достоинств и недостатков каждого алгоритмы. В результате сделаны выводы и определены два алгоритма, которые считаются наилучшими, и внесены предложения по созданию новых алгоритмов, без недостатков в уже существующих. Приведен практический эксперимент для доказательства того, что новые алгоритмы работают лучше чем существующие.

The article discusses s problem of a plagiarism. At the beginning describes the models of program representation, and algorithms for determining plagiarism in software code C #. Each algorithm was analyzed and compared to determine the strengths and weaknesses. As a result, conclusions were made and identified two algorithms, which are considered the best, and made proposals to develop new algorithms without the disadvantages of existing ones. The practical experiment was held to prove that the new algorithms work better than existing ones.

Введение

Возможность легкого копирования информации, представленной в электронном виде, породила множество проблем, связанных с нарушением авторских прав. В бумажных и электронных изданиях часто можно встретить публикации об очередном противостоянии пиратов и владельцев авторских прав на распространяемые этими пиратами произведения. Как правило, в таких случаях единственным (но не всегда действенным) способом улаживания конфликтов являются судебные разбирательства.

Сейчас в век интенсивного развития информационных технологий интеллектуальная собственность становиться все более ценной. В связи со значительным увеличением объемов этого вида собственности назрела необходимость в мощных автоматических инструментах для защиты авторских прав, для инспектирования и проверки авторства, для нахождения плагиата.

- Плагиат – вид нарушения авторских прав. Принуждение к соавторству также рассматривается как плагиат [1].

Виды представления исходных кодов программ

В виде элемента n-мерного пространства. Ранние системы по обнаружению плагиата (например,[2]) представляли программу, как точку в n-мерном пространстве натуральных чисел с нулем, i-ая координата которой - количественная характеристика какого-либо свойства (атрибута) всей программы. Например, средняя длина строки кода, количество объявленных и используемых переменных, средняя длина имен переменных, количество операторов ветвления и так далее. Если точки двух программ лежат рядом, то одна из них предполагается плагиатом другой.

Исходный код

Некоторые системы обнаружения плагиата рассматривают исходный код “как есть”. Например, так поступают детекторы плагиата, которые работают с кодом так же, как и с обычными текстами. Но они крайне неэффективны для решения нашей задачи, так как переименование функций и переменных или несущественные изменения в коде являются серьезными препятствиями для их правильной работы. Иногда используется параметризованное представление кода (см. [3]).

Токенизация

Пусть у нас есть две строки кода f or(int i = 0; i <= n; i + +) и f or(int j = 0; (j 1) < n; j + +). Очевидно, что у них одинаковая функциональность, и плагиатор мог из одной получить

другую без особых усилий, а для ранее описанных представлений они совершенно различны. Чтобы бороться с такого рода средствами сокрытия плагиата было придумано токенизированное представление кода. Основная идея этого представления это сохранение существенных и игнорирование всех поверхностных (то есть легко модифицируемых) деталей кода программы. Процедура токенизации выглядит примерно так:

1) Каждому оператору языка (кроме пустого его игнорируем), который не является операндом, приписываем код, назначенный заранее для каждого класса операторов. Также коды можно приписывать блочным операторам (наприме, begin/end, {}), подключениям библиотек и заголовочных файлов.

2) Строим строку из полученных кодов, сохраняя порядок следования их в исходном коде программы. Один символ строки (токен) код одного оператора.

Таким образом мы автоматически игнорируем названия функций и переменных (классов, объектов и так далее), разделительные символы, предотвращаем влияние мелких изменений кода программы.

Нужно отметить, что процесс токенизации и разбиение операторов на классы зависим от используемого в исходном коде языка программирования.

К одному классу операторов обычно относят те, которым соответствует один идентификатор языка программирования, все вызовы функций, вызовы методов классов, объявления переменных элементарных типов, объявление экземпляров классов. Более подробную информацию можно посмотреть в приложении к [4].

Анализ существующих алгоритмов

В данном алгоритме над кодом программы выполняется токенизация в результате получается строка токенов, которая впослідствии разбиваются на k-граммы, строится таблица в которой записан каждый k-грамм, а так же его частота встречаемости в строке, чем меньше частота встречаемости, тем больше вес данного k-грамма [5]. По частоте встречаемости высчитывается процент каждого k-грамма в коде.

При сравнении находятся все совпавшие k-граммы и в программе из базы суммируются проценты данных k-граммов в результате выводится процент схожести.

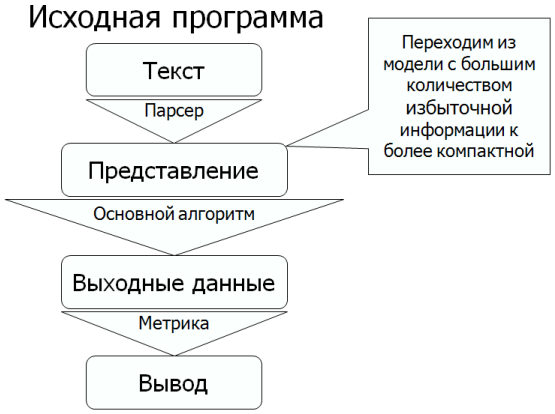

Рис.1. Общая схема поиска плагиата.

Алгоритм Хескела

Достоинства:

1) Линейная трудоемкость алгоритма

Недостатки:

2) Возможность совпадения токенизированного представления программ, но отсутствия совпадения в исходных кодах программ.

3) Небольшое количество уникальных k-граммов в больших программах, соответственно многие совпадения, не содержащие в себе таких k-граммов будут проигнорированы.

4) Вставка в найденный блок или изменение на семантически эквивалентный оператора во многих случаях будет приводить к игнорированию той части блока, в которой не содержится уникальный k-грамм.

5) Нельзя организовать базу данных, ускоряющую проверку один- против-всех.

Метод отпечатков

В этом алгоритме токенезированная программа представляется в виде набора отпечатков (меток, fingerprints), так чтобы эти наборы для похожих программ пересекались. Этот метод позволяет организовать эффективную проверку по базе данных. Метод отпечатков можно представить в виде четырех нижеследующих шагов:

1) Последовательно хэшируем подстроки токенизированной программы P длины k (фиксированный параметр).

2) Выделяем некоторое подмножество их хэш-значений, хорошо характеризующее P. Проделываем те же шаги для токенизированных программ $T_1, T_2 ... T_n$ и помещаем их выбранные хэш-значения в хэш-таблицу.

3) С помощью хэш-таблицы (базы) получаем набор участков строки P , подозрительных на плагиат.

4) Анализируем полученные на предыдущем шаге данные и делаем выводы.

Существует несколько реализаций шага 2, но наиболее качественный это использование метода просеивания [6].

Достоинства:

1) Можно организовать базу данных, ускоряющую проверку один-против-всех.

2) Преимущества токенизированного представления.

3) Общие подстроки, меньше пороговой длины игнорируются, поэтому алгоритм не принимает в расчет малые случайно совпавшие участки кода.

4) При разбиении совпавшего участка кода на две и более части вставкой одного-нескольких блоков или одиночных операторов, а также перестановкой небольшого количества независимых операторов, функция схожести слабо изменяется. (Длина совпадения должна быть значительно больше некоторой пороговой длины.)

5) Алгоритм нечувствителен к перестановкам больших фрагментов кода.

Недостатки:

1) Возможность совпадения токенизированного представления программ, но отсутствия совпадения в исходных кодах программ.

2) Разбиение совпадения на блоки, вставкой или заменой оператора на похожий (например, for на while), каждый длиной меньше k, ведет к полному игнорированию совпадения.

Алгоритм выравнивания строк

Пусть у нас есть две программы, представим их в виде строк токенов s и t соответственно (возможно различной длины). Теперь мы можем воспользоваться методом локального выравнивания строк, разработанным для определения схожести строк ДНК [7]. Выравнивание двух строк получается с помощью вставки в них пробелов таким образом, чтобы их длины стали одинаковыми. Заметим, что существует большое количество различных выравниваний двух строк.

Достоинства:

1) Использование процедуры кластеризации.

2) Токенизированное представление программ

Недостатки:

1) На базе этого алгоритма нельзя организовать базу данных, ускоряющую проверку один-против-всех.

Алгоритм жадного строкового замощение

Эвристический алгоритм получения жадного строкового замощения (The Greedy String Tiling, см. [8]), получает на вход две строки символов над определенным алфавитом (у нас это множество допустимых токенов), а на выходе дает набор их общих непересекающихся подстрок близкий к оптимальному.

Достоинства:

1) Преимущества токенизированного представления.

2) Общие подстроки меньшей длины, чем MinimumMatchLength игнорируются, поэтому алгоритм не принимает в расчет небольшие случайно совпавшие участки кода.

3) При разбиении совпавшего участка кода на две и более части вставкой одного-нескольких блоков или одиночных операторов, а также перестановкой небольшого количества независимых операторов, функция схожести слабо изменяется.

4) Алгоритм нечувствителен к перестановкам больших фрагментов кода.

Недостатки:

1) Возможноть совпадения токенизированного представления программ, но отсутствия совпадения в исходных кодах программ.

2) Разбиение совпадения на блоки, вставкой или заменой оператора на похожий (например, for на while), каждый длиной меньше MinimumMatchLength, ведет к полному игнорированию совпадения.

3) Из-за эвристик, используемых в алгоритме, совпадения, длиной меньшей чем MinimumMatchLength, будут проигнорированы.

4) Нельзя организовать базу данных, ускоряющую проверку один-против-всех.

Выводы

Наибольшим количеством достоинств является «Метод отпечатков», который разрешает поддержку полноценной базы большого размера, позволяющую ускорить процедуру провер-

ки один-против-всех, так же относительно данной базы можно искать плагиат в любом другом исходном коде за приемлемое время, а не проверять образец с каждым элементом базы, это происходит из-за того, что этот алгоритм использует хеш-таблицы.

Все описанные алгоритмы, кроме метода отпечатков, не могут эффективно работать с большой базой, так как требуют проведения сравнений образца с каждым элементом базы.

Алгоритм Хескела имеет множество недостатков относительно метода отпечатков и метода жадного строкового замощения, потому, что производит сравнение по k-граммам, а не по хеш значениям, однако если применить данный алгоритм к уже полученным меткам в методе отпечатков, то можно ускорить процесс сравнения.

Предложения по улучшению рассмотренных алгоритмов

При выполнении токенизации, ввести одинаковые коды на схожие операторы, например for и while или double и float, что бы соответствовали одному коду токенизации. Далее ввести понятие вложенный токен, например если у нас есть целочисленная функция Test, которая содержит в себе цикл for и который в свою очередь содержит переменные типа int и float.

```
int Test ()
{
    for
   {
     int x;
     float y;
    }
}
```

И например целочисленной переменной соответствует токен a, а циклу for токен b и переменным int и float соответственно токены c и d, то вложенный токен будет иметь вид: a{b{cd}}, который при указании k-грамма будет приравниваться одному символу, например если k=3, то он будет иметь вид (рисунок 2):

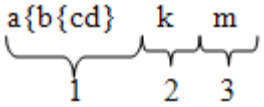

Рис. 2. Схема вложенного токина.

где a{b{cd}} – первый символ, k-второй, m-третий.

Так же предлагается преобразовать метод просеивания который используется для получения меток из хеш значений, что бы уже когда создано окно для поиске меток, то если в нем имеются хеш значения, полученные из внутреннего токена, то они обязательно становятся метками, а уже если таких значений нет, то выбирается наименьшее значение в окне, данный подход дожжен позволить находить частичный плагиат, например одинаковые функции. Когда будут получены метки, то они сохраняются в таблице поиска типа:

Табл. 2. Таблица поиска

Метка	k-грамм	Имя файла	строка	Метод (цикл)
12 4	3	Code.txt .	5 .	+ -

Когда будет производится сравнение на плагиат, то сравнимая программа которая имеет свой набор меток, будет сверятся с каждой меткой в таблице поиска, так можно будет сделать вывод в процентном соотношении к какому файлу она наиболее близка. Последний столбец Метод (цикл) существует для того, что бы показать что в данной метке находится вложенный токен, который изначально имеет также свое хеш значение, но оно хранится в таблице сравнения , например если мы имеем k-грамм a{b{cd}}k m и он например имеет значение 12, то мы видим что в данном k-грамме есть вложенный токен, которому отдельно выдается хеш значение, например 8 которое записывается в таблицу сравнения, которая будет иметь вид:

Табл. 3. Таблица Сравнения

Хеш значение вложенного точена	Хеш значеие всего k-грамма
8	12

Это делается для того, что бы даже если у k-граммов которые имеют разные значения хеш функций, но имеют одинаковый вложенный токен было найдено по нему совпадение.

Предложенные Алгоритмы

Вариант 1:

1) Производим токенизацию, так же создаются вложенные токены.

2) Выполняем разбиение на K-граммы (n = m-(k-1), где n- количество K-грамм, m- длинна текста).

3) Получаем хеш значения с помощью хеш-функции для каждого K-грамма, а также для каждого вложенного токена.

4) Используем метод просеивания для выбора меток из всех существующих хеш значений (выбираем окно размерностью w=t–k+1, если в данном окне есть хеш значение вложенного токена, то оно обязательно становится меткой, если же таких значений нет, то в окне выбираем найменьшое хеш значение, которое становиться меткой, сравниваем все метки, и если есть метки с одинаковыми значениями, то они записываются в таблицы только один раз, метки полученные от K-граммов записываются в таблицу поиска, а метки от вложенных токенов в таблицу сравнения.)

Вариант 2:

1) Производим токенизацию, так же создаются вложенные токены.

2) Выполняем разбиение на K-граммы (n = m-(k-1), где n- количество K-грамм, m- длинна текста).

3) Получаем хеш значения с помощью хеш-функции для каждого K-грамма, а также для каждого вложенного токена.

4) Используем метод просеивания для выбора меток из всех существующих хеш значений (выбираем окно размерностью w=t–k+1, если в данном окне есть хеш значение вложенного токена, то оно обязательно становится меткой, если же таких значений нет, то в окне выбираем найменьшое хеш значение, которое становиться меткой, метки полученные от K-граммов записываются в таблицу поиска, а метки от вложенного токена в таблицу сравнения.)

5) Используем метод Хескела к полученным меткам, после чего создается таблица, где каждой метке указывается ее вес (чем меньше раз данная метка встречается в таблице поиска и в таблице сравнения, тем больше ее вес).

На рисунке 3 представлена Диаграмма классов для разрабатываемой системы

Анализ предложенных вариантов

В данных вариантах предлагается ввести понятие вложенный токен, что позволит находить плагиат отдельных функций в кодах программ. Введение вложенных токенов влечет за собой модификацию метода просеивания, в котором изменяется способ отбора меток в окне. Первый вариант алгоритма должен работать быстрее второго, так как во втором варианте будет строится еще и третья таблица для метода Хескела. За счет уменьшения быстродействия должно увеличится качество поиска относительно первого варианта.

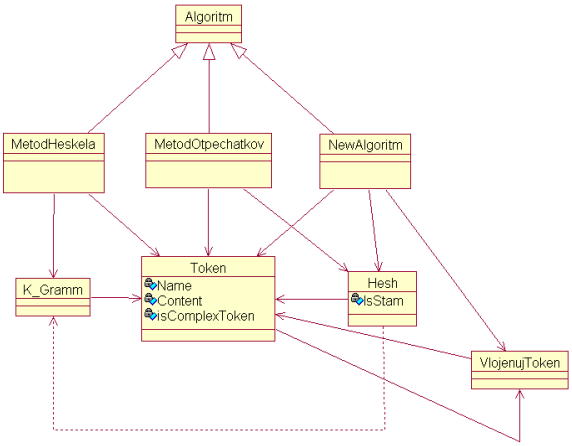

Рис. 2. Диаграмма классов

Экспериментальный анализ работы алгоритмов

Проведем экспериментальный анализ описанных выше алгоритмов, целью которого является поиск скрытого плагиата.

Анализ работы алгоритмов средствами ПК

В эксперименте используем несколько идентичных проектов на языке C# платформы .NET framework компании Microsoft (в данном эксперименте используются три проекта с приблизительным количеством строк кода равным 2150).

Один из проектов оставляем без изменений, в остальных моделируем некоторые попытки скрыть плагиат:

- замена названий переменных (методов);
- изменение последовательности чередования методов (функций), переменных;
- изменение типов объявления переменных;
- изменение типов объявления циклов.

В таблице 4 представлен результат сравнения проектов разными алгоритмами поиска плагиата.

На рисунке 3 полученные данные представлены в виде точечной диаграммы.

Как видно из полученных результатов, лучшие показатели поиска плагиата показал алгоритм описан в варианте 1 (Z-nested 1).

Табл. 4. Результаты сравнения проектов разными алгоритмами поиска плагиата

№	Файл проекта с базы	Алго-ритм	Время срав-нения (мс)	Похожесть (%)
1	CodePlagiat2.csproj	Z-nested 1	661	100
2	CodePlagiat.csproj	Z-nested 1	498	96,4285714285714
3	CodePlagiat1.csproj	Z-nested 1	368	95,7908163265306
4	CodePlagiat2.csproj	Heskel	99	100
5	CodePlagiat.csproj	Heskel	670	78,5714285714289
6	CodePlagiat1.csproj	Heskel	193	77,9038718291058
7	CodePlagiat2.csproj	Z-nested 2	519	100
8	CodePlagiat.csproj	Z-nested 2	263	83,9080459770115
9	CodePlagiat1.csproj	Z-nested 2	69	85,0574712643678

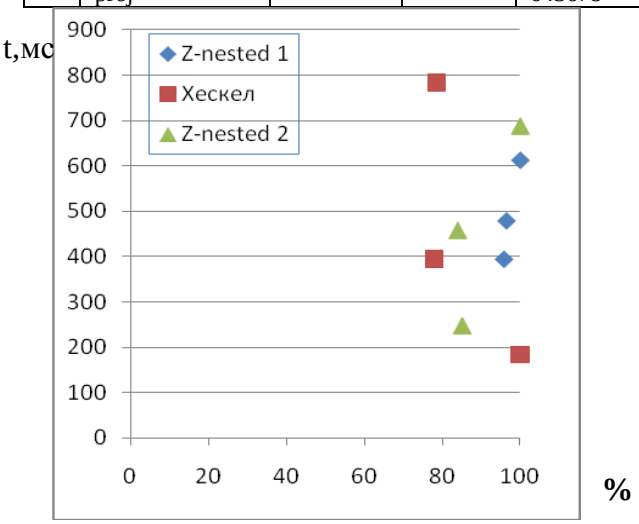

Рис. 3. Точечная диаграмма сравнения алгоритмов поиска плагиата

Анализ работы алгоритмов пошаговым расчетом

Представлено подробное описание работы алгоритмов с указанными примерами.

Алгоритм Хескела

Рассмотрим на приведенных примерах программ работу алгоритма Хескела.

1)Строим строку проекта в виде последовательности токенов.

Пример №1.

080|011|080|004|076|080|060|037|060|017|081|076|080|034|077|080|029|033|080|060|081|081|081|081|081

Пример №2.

080|011|080|004|076|080|034|077|080|029|033|080|060|081|081|081|076|080|060|037|060|017|081|081|081

Пример №3.

080|011|080|004|076|080|034|029|033|080|028|082|080|060|081|081|081|076|080|060|037|060|017|081|081|081

Пример №4.

080|011|080|004|076|080|029|033|080|028|082|080|060|081|081|034|081|076|080|060|037|060|017|081|081|081

Далее выбираем проект, который хотим сравнить с остальными. Допустим, это будет проект Пример №1 (далее основной проект).

2) Строим последовательность k-граммов каждого проекта с учетом количества их встречаемости. Допустим, количество токенов в одном к-грамме равно 3. Тогда имеем следующее:

Табл. 4. Пример №1.

№	К-грамм	Частота встреча-емости	Вес (%)
1	080|011|080	1	4,34782608695652
2	011|080|004	1	4,34782608695652
3	080|004|076	1	4,34782608695652
4	004|076|080	1	4,34782608695652
5	076|080|060	1	4,34782608695652
6	080|060|037	1	4,34782608695652
7	060|037|060	1	4,34782608695652
8	037|060|017	1	4,34782608695652
9	060|017|081	1	4,34782608695652
10	017|081|076	1	4,34782608695652
11	081|076|080	1	4,34782608695652
12	076|080|034	1	4,34782608695652
13	080|034|077	1	4,34782608695652
14	034|077|080	1	4,34782608695652
15	077|080|029	1	4,34782608695652
16	080|029|033	1	4,34782608695652
17	029|033|080	1	4,34782608695652
18	033|080|060	1	4,34782608695652
19	080|060|081	1	4,34782608695652
20	060|081|081	1	4,34782608695652
21	081|081|081	3	13,0434782608696

Табл. 5. Пример №2.

№	К-грамм	Частота встреча-емости	Вес (%)
1	080|011|080	1	4,34782608695652
2	011|080|004	1	4,34782608695652
3	080|004|076	1	4,34782608695652
4	004|076|080	1	4,34782608695652
5	076|080|034	1	4,34782608695652
6	080|034|077	1	4,34782608695652
7	034|077|080	1	4,34782608695652
8	077|080|029	1	4,34782608695652
9	080|029|033	1	4,34782608695652

10	029|033|080	1	4,34782608695652
11	033|080|060	1	4,34782608695652
12	080|060|081	1	4,34782608695652
13	060|081|081	1	4,34782608695652
14	081|081|081	2	8,69565217391304
15	081|081|076	1	4,34782608695652
16	081|076|080	1	4,34782608695652
17	076|080|060	1	4,34782608695652
18	080|060|037	1	4,34782608695652
19	060|037|060	1	4,34782608695652
20	037|060|017	1	4,34782608695652
21	060|017|081	1	4,34782608695652
22	017|081|081	1	4,34782608695652

Табл. 5. Пример №3

№	К-грамм	Частота встречаемости	Вес (%)
1	080|011|080	1	4,16666666666667
2	011|080|004	1	4,16666666666667
3	080|004|076	1	4,16666666666667
4	004|076|080	1	4,16666666666667
5	076|080|034	1	4,16666666666667
6	080|034|029	1	4,16666666666667
7	034|029|033	1	4,16666666666667
8	029|033|080	1	4,16666666666667
9	033|080|028	1	4,16666666666667
10	080|028|082	1	4,16666666666667
11	028|082|080	1	4,16666666666667
12	082|080|060	1	4,16666666666667
13	080|060|081	1	4,16666666666667
14	060|081|081	1	4,16666666666667
15	081|081|081	2	8,33333333333333
16	081|081|076	1	4,16666666666667
17	081|076|080	1	4,16666666666667
18	076|080|060	1	4,16666666666667
19	080|060|037	1	4,16666666666667
20	060|037|060	1	4,16666666666667
21	037|060|017	1	4,16666666666667
22	060|017|081	1	4,16666666666667
23	017|081|081	1	4,16666666666667

Табл. 6. Пример №4

№	К-грамм	Частота встречаемости	Вес (%)
1	080|011|080	1	4,16666666666667
2	011|080|004	1	4,16666666666667
3	080|004|076	1	4,16666666666667
4	004|076|080	1	4,16666666666667
5	076|080|029	1	4,16666666666667
6	080|029|033	1	4,16666666666667
7	029|033|080	1	4,16666666666667
8	033|080|028	1	4,16666666666667
9	080|028|082	1	4,16666666666667
10	028|082|080	1	4,16666666666667
11	082|080|060	1	4,16666666666667
12	080|060|081	1	4,16666666666667
13	060|081|081	1	4,16666666666667
14	081|081|034	1	4,16666666666667
15	081|034|081	1	4,16666666666667
16	034|081|076	1	4,16666666666667
17	081|076|080	1	4,16666666666667
18	076|080|060	1	4,16666666666667
19	080|060|037	1	4,16666666666667
20	060|037|060	1	4,16666666666667
21	037|060|017	1	4,16666666666667
22	060|017|081	1	4,16666666666667
23	017|081|081	1	4,16666666666667
24	081|081|081	1	4,16666666666667

3)Основываясь на описанных выше таблицах выполняем сравнение проектов. Если k-грамм присутствует в основном и в сравниваемом проектах, то учитывается его вес в %, который он занимает в основном. Далее в таблице приведен результат сравнения.

Табл. 7. Результат сравнения

№	Основной проект	Сравниваемый проект	Процент схожести (%)
1	Пример №1	Пример №2	91,304347826087
2	Пример №1	Пример №3	66,6666666666667
3	Пример №1	Пример №4	62,5

Алгоритм Z-nested 1

Рассмотрим на приведенных примерах программ работу алгоритма описанного в варианте №1 (Z-nested 1).

1) Переводим проект в последовательность токенов. Далее получаем хеш значения к-граммов. Допустим, размер к-грамма равен 3, тогда имеем следующий вид проектов:

Табл. 8. Пример №1

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	378461068	011|080|004
3	-390511174	080|004|076{080060060081}
4	1776990261	004|076{080060060081}|028{080060081}
5	95065247	076{080060060081}|028{080060081}|080028{080060081}081
6	-1505237569	028{080060081}|080028{080060081}081|028{080028{080060081}081}
7	-1005543780	080028{080060081}081|028{080028{080060081}081}|080034028{080028{080060081}081}081
8	-1407541811	028{080028{080060081}081}|080034028{080028{080060081}081}0

		81|076{080034028{080028{08006 0081}081}081}
9	-499040212	080034028{080028{080060081}08 1}081|076{080034028{080028{08 0060081}081}081}|081

Табл. 8. Пример №2.

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	378461068	011|080|004
3	-1906288165	080|004|028{080060081}
4	-823501287	004|028{080060081}|080028{080 060081}081
5	-1505237569	028{080060081}|080028{0800600 81}081|028{080028{080060081}0 81}
6	-1005543780	080028{080060081}081|028{0800 28{080060081}081}|080034028{0 80028{080060081}081}081
7	-1407541811	028{080028{080060081}081}|080 034028{080028{080060081}081} 081|076{080034028{080028{0800 60081}081}081}
8	-1573902207	080034028{080028{080060081}0 81}081|076{080034028{080028{0 80060081}081}081}|076{0800600 60081}
9	783746616	076{080034028{080028{0800600 81}081}081}|076{080060060081} |081

Табл. 9. Пример №3.

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	378461068	011|080|004
3	-1906288165	080|004|028{080060081}
4	-823501287	004|028{080060081}|080028{0800 60081}081
5	-1505237569	028{080060081}|080028{0800600 81}081|028{080028{080060081}0 81}
6	-1005543780	080028{080060081}081|028{0800 28{080060081}081}|080034028{0 80028{080060081}081}081
7	-1407541811	028{080028{080060081}081}|080 034028{080028{080060081}081}0 81|076{080034028{080028{08006 0081}081}081}
8	-1573902207	080034028{080028{080060081}08 1}081|076{080034028{080028{08
		0060081}081}081}|076{08006006 0081}
9	783746616	076{080034028{080028{08006008 1}081}081}|076{080060060081}|0 81

Табл. 10. Пример №4

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	378461068	011|080|004
3	-1906288165	080|004|028{080060081}
4	-823501287	004|028{080060081}|080028{080 060081}081
5	-1505237569	028{080060081}|080028{080060 081}081|028{080028{080060081 }081}
6	-1555898206	080028{080060081}081|028{080 028{080060081}081}|080028{08 0028{080060081}081}034081
7	670774381	028{080028{080060081}081}|08 0028{080028{080060081}081}03 4081|076{080028{080028{08006 0081}081}034081}
8	-1386322015	080028{080028{080060081}081} 034081|076{080028{080028{080 060081}081}034081}|076{08006 0060081}
9	-144113634	076{080028{080028{080060081} 081}034081}|076{080060060081 }|081

2) Используем метод просеивания для выбора меток из всех существующих хеш значений к-граммов. Допустим, окно w используемое в алгоритме равно 2, тогда имеем следующие метки проектов:

Табл. 11. Пример №1

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	-390511174	080|004|076{080060060081}
3	1776990261	004|076{080060060081}|028{080 060081}
4	95065247	076{080060060081}|028{080060 081}|080028{080060081}081
5	-1505237569	028{080060081}|080028{080060 081}081|028{080028{080060081 }081}
6	-1005543780	080028{080060081}081|028{080 028{080060081}081}|080034028 {080028{080060081}081}081
7	-1407541811	028{080028{080060081}081}|08 0034028{080028{080060081}081 }081|076{080034028{080028{08 0060081}081}081}
8	-499040212	080034028{080028{080060081}0

		81}081|076{080034028{080028{080060081}081}081}|081

Табл. 12. Пример №2

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	-1906288165	080|004|028{080060081}
3	-823501287	004|028{080060081}|080028{080060081}081
4	-1505237569	028{080060081}|080028{080060081}081|028{080028{080060081}081}
5	-1005543780	080028{080060081}081|028{080028{080060081}081}|080034028{080028{080060081}081}081
6	-1407541811	028{080028{080060081}081}|080034028{080028{080060081}081}081|076{080034028{080028{080060081}081}081}
7	-1573902207	080034028{080028{080060081}081}081|076{080034028{080028{080060081}081}081}|076{080060060081}
8	783746616	076{080034028{080028{080060081}081}081}|076{080060060081}|081

Табл. 13. Пример №3

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	-1906288165	080|004|028{080060081}
3	-823501287	004|028{080060081}|080028{080060081}081
4	-1505237569	028{080060081}|080028{080060081}081|028{080028{080060081}081}
5	-1005543780	080028{080060081}081|028{080028{080060081}081}|080034028{080028{080060081}081}081
6	-1407541811	028{080028{080060081}081}|080034028{080028{080060081}081}081|076{080034028{080028{080060081}081}081}
7	-1573902207	080034028{080028{080060081}081}081|076{080034028{080028{080060081}081}081}|076{080060060081}
8	783746616	076{080034028{080028{080060081}081}081}|076{080060060081}|081

Табл. 14. Пример №4

№	Хеш-значение к-грамма	К-грамм
1	-1494513560	080|011|080
2	-1906288165	080|004|028{080060081}
3	-823501287	004|028{080060081}|080028{080060081}081
4	-1505237569	028{080060081}|080028{080060081}081|028{080028{080060081}081}
5	-1555898206	080028{080060081}081|028{080028{080060081}081}|080028{080028{080060081}081}034081
6	670774381	028{080028{080060081}081}|080028{080028{080060081}081}034081|076{080028{080028{080060081}081}034081}
7	-1386322015	080028{080028{080060081}081}034081|076{080028{080028{080060081}081}034081}|076{080060060081}
8	-144113634	076{080028{080028{080060081}081}034081}|076{080060060081}|081

Далее выбираем проект, который хотим сравнить с остальными. Допустим это будет проект Пример №1 (далее основной проект).

3) При сравнении основного проекта с сравниваемыми учитываются вложенные к-граммы. Т.е., если хеш-значения к-граммов не совпадают, но при этом вложенные токены к-граммов одинаковые, то к-граммы считаются одинаковыми.

Табл. 15. Результат сравнения

№	Основной проект	Сравниваемый проект	Процент схожести (%)
1	Пример №1	Пример №2	100
2	Пример №1	Пример №3	100
3	Пример №1	Пример №4	87,5

Алгоритм Z-nested 2

Рассмотрим на приведенных примерах программ работу алгоритма описанного в варианте №2 (Z-nested 2).

Пункты 1 и 2 аналогичны описанным в работе алгоритма **Z-nested 1.**

3) Далее применяем алгоритм Хескела к полученным меткам. Допустим, количество меток в одном к-грамме равно 3. Тогда имеем следующие значения к-граммов построенных из меток:

Табл. 16. Пример №1

№	К-грамм меток хеш значений	К-грамм токенов хеш значений
1	-1494513560|-390511174|1776990261	080|011|080*080|004|076{080060060081}*004|076{080060060081}|028{080060081}

2	95065247|-1505237569|-1005543780	076{080060060081}|028{080060081}|080028{080060081}081*028{080060081}|080028{080060081}081|028{080028{080060081}081}*080028{080060081}081|028{080028{080060081}081}|080034028{080028{080060081}081}081

Табл. 17. Пример №2

№	К-грамм меток хеш значений	К-грамм токенов хеш значений
1	-1494513560|-1906288165|-823501287	080|011|080*080|004|028{080060081}*004|028{080060081}|080028{080060081}081
2	-1505237569|-1005543780|-1407541811	028{080060081}|080028{080060081}081|028{080028{080060081}081}*080028{080060081}081|028{080028{080060081}081}|080034028{080028{080060081}081}081*028{080028{080060081}081}|080034028{080028{080060081}081}081|076{080034028{080028{080060081}081}081}

Табл. 18. Пример №3

№	К-грамм меток хеш значений	К-грамм токенов хеш значений
1	-1494513560|-1906288165|-823501287	080|011|080*080|004|028{080060081}*004|028{080060081}|080028{080060081}081
2	-1505237569|-1005543780|-1407541811	028{080060081}|080028{080060081}081|028{080028{080060081}081}*080028{080060081}081|028{080028{080060081}081}|080034028{080028{080060081}081}081*028{080028{080060081}081}|080034028{080028{080060081}081}081|076{080034028{080028{080060081}081}081}

Табл. 19. Пример №4

№	К-грамм меток хеш значений	К-грамм токенов хеш значений
1	-1494513560|-1906288165|-823501287	080|011|080*080|004|028{080060081}*004|028{080060081}|080028{080060081}081
2	-1505237569|-1555898206|670774381	028{080060081}|080028{080060081}081|028{080028{080060081}081}*080028{080060081}081|028{080028{080060081}081}|080028{080028{080060081}081}034081*028{080028{080060081}081}|080028{080028{080060081}081}034081|076{080028{080028{080060081}081}034081}

4)При сравнении основного проекта с сравниваемыми учитываются вложенные токены. Т.е., если к-граммы разные, но при этом вложенные токены к-граммов одинаковые, то к-граммы считаются одинаковыми.

Табл. 20. Результат сравнения

№п.п	Основной проект	Сравниваемый проект	Процент схожести (%)
1	Пример №1	Пример №2	100
2	Пример №1	Пример №3	100
3	Пример №1	Пример №4	100

Список литературы

1. Большой юридический словарь Под ред. А.Я. Сухарева, В.Е. Крутских., М., 2002
2. Faidhi, J. A. W. and S. K. Robinson. An Empirical Approach for Detecting Program Similarity within a University Programming Environment. Computers and Education 11(1): pp. 1119 (1987).
3. B. S. Baker. On Finding Duplication and Near-Duplication in Large Software Systems. In Proceedings of the second IEEE Working Conference on Reverse Engineering (WCRE), July 1995, pp. 86–95.
4. L. Prechelt, G. Malpohl, and M. Philippsen. JPlag: Finding plagiarisms among a set of programs. Technical Report No. 1/00, University of Karlsruhe, Department of Informatics, March 2000.
5. Heckel, Paul. A Technique for Isolating Di erences Between Files. Communications of the ACM 21(4), pp. 264–268 (April 1978).
6. Aiken, S. Schleimer, D. Wikerson. Winnowing: local algorithms for document fingerprinting. In Pro ceedings of ACMSIGMOD Int. Conference on Management of Data, San Diego, CA, June 9–12, pp. 76–85. ACM Press, New York, USA, 2003.
7. X. Huang, R. C. Hardison, AND W. Miller. A space-e cient algorithm for local similarities. Computer Applications in the Biosciences, 6 (1990), pp. 373-381. Michael J. Wise. String similarity via greedy string tiling and running Karp-Rabin matching. Dept. of CS, University of Sydney, December 1993.