*МРНТИ 50.05.03*

*Е. Амиргалиев*[1], *С. Бегадил*[1]
[1]Университет имени Сулеймана Демиреля
г. Алматы, Казахстан

# ПОИСК ДУБЛИКАТОВ В ТЕКСТЕ С ИСПОЛЬЗОВАНИЕМ АЛГОРИТМА ШИНГЛА

**Аннотация.** В этой работе представлен подход анализа текстовой информации по задаче поиска дубликатов в тесте на основе алгоритма Шингла. Алгоритм реализован программным путем на контрольных примерах. Оценен результат алгоритма.

**Ключевые слова**: дубликат, неточный дубликат, шингл, хеши, супершингл.

***

**Аңдатпа.** Мақалада мәтіндік ақпаратты талдау барысында Шингл алгоритміне негізделген ұқсас мәтіндерді табу есебі қарастырылған. Алгоритм бақылау мысалдары үшін бағдарламалық жолмен іске асырылған. Алгоритм нәтижесі бағаланған.

**Кілт сөздер.** Дубликат, дәл емес дубликат, шингл, хеш, супершингл.

***

**Abstract.** In this paper, we present an approach for analyzing test information on the task of finding duplicates in a test based on the Shingle algorithm. The algorithm is implemented programmatically on test cases. The result of the algorithm is estimated.

**Keywords:** duplicate, inaccurate duplicate, shingle, hash, super-shingle.

В современных условиях проблема анализа и оценки всевозрастающего объема текстовой информации и изображений является актуальной задачей. Естественно, с усложнением представления изображении и большего объема текстовой информации, особенно, если в тексте встречаются слова и предложения на разных языках в процессе анализа поиск дубликатов (плагиат) является одной из важных и сложных задач. Задачи повышения качества анализа и определения дубликатов остается важнейшей задачей в компьютерном анализе. Основная проблема решения этой задачи - огромный объем данных, в том числе неструктурированных. Огромный объем данных делает его прямое решение (путем сравнения текстов документов) практически невозможным в разумные сроки.

В этой работе мы представили один из подходов решения данной задачи с помощью реализации алгоритма Шингла на контрольных примерах. Используя этот алгоритм, мы в расширенном варианте можем решать такие задачи, как поиск похожих записей [1, 2], поиск похожих изображений, группировка похожих документов и т.д.

*Введение*

Поиск неточных дубликатов позволяет предположить, являются ли два объекта частично идентичными или нет. Объект можно понимать как текстовые файлы, веб-страницы и другие типы данных. В этой статье я буду работать с текстом, но после понимания того, как работает алгоритм, перенос реализации на объекты, которые нам нужны, не составит труда.

В публикации об алгоритмах поиска неточных дубликатов используется версия алгоритма шингла, в которой используется случайная выборка из 84 случайных шинглова. Почему 84? Использование 84 случайно выбранных значений контрольной суммы позволит легко модифицировать алгоритм алгоритма супер-шинглы и мега-шинглы, которые значительно менее ресурсоемкими [3]. Вполне возможно использовать один алгоритм для вычисления контрольной суммы, и этого будет достаточно для небольшого количества документов, но для минимизации числа операций сравнения нам нужно использовать 84 случайно выбранных контрольных сумм. Я использовал только один хэш CRC32 в моей реализации, но мы можем использовать другие хеши, напримерSHA1, MD5 и т.д.

*Алгоритм шингла*

Например, у нас есть два текста, и нам нужно предположить, являются ли они почти дубликатами. Реализацияалгоритмавключает в себянесколькоэтапов:

- канонизациятекста;
- разделениетекстанашинглы;
- контрольныесуммырасчетов;
- поискидентичныхподпоследовательностей.

*Канонизация текста.* Контрольные суммы очень чувствительны к изменениям. Например, контрольные суммы для следующих двух текстов будут разными.

Hello world : -1948869038

Hello world! : 461707669

Разница между первым и вторым текстом только в символе в конце текста, но, как мы видим, контрольные суммы совершенно разные. Когда мы ищем дубликаты, нас не интересуют никакие восклицательные знаки, точки, запятые и т. Д. Мы заинтересованы только в словах (а не союзах, предлогах и т.д.).

Поэтому нам нужно очистить текст от бесполезных знаков и слов, которые не имеют смысла при сравнении, этот процесс называется «канонизация текста». Прежде всего нужно создать набор символов и слов, которые необходимо исключить из обоих текстов. Затем нужно очистить тексты от ненужных символов и слов. В рассмотренном варианте наборы стоп-слов и стоп-символов выглядят следующим образом:

стоп слова: слова, такие как я, я, мой, я, мы, наш, наш, мы, он, он и т. д. [2].

стопсимволы: .,!?:; - ()

К стоп символам включены знаки препинания, символы новой строки и вкладки. К стоп словам я включал союзы, предлоги и другие слова, которые при сравнении не должны влиять на результат. В общем, преобразование текстов в одну каноническую форму не ограничивается описанными действиями. Например, можно использовать морфологические анализаторы языков и т. д.

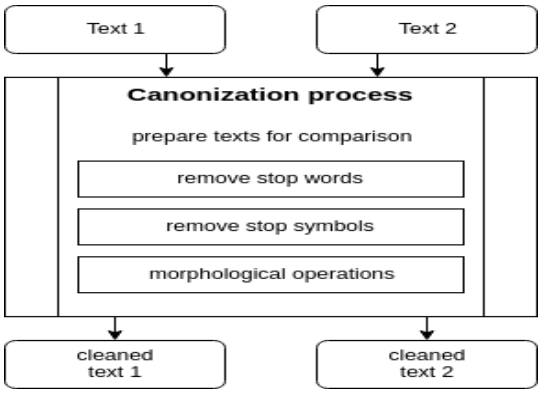


Рис.1. Канонизация текстов

*Разделение текста на шинглы*. После этапа канонизации нам нужно разбить тексты на подпоследовательности - шинглов. В моем примере я использовал 3 слова. Таким образом, разбивая текст на подпоследовательности, мы получаем набор шинглов. Количество установленных равняется количеству слов минус длина шингла плюс один.

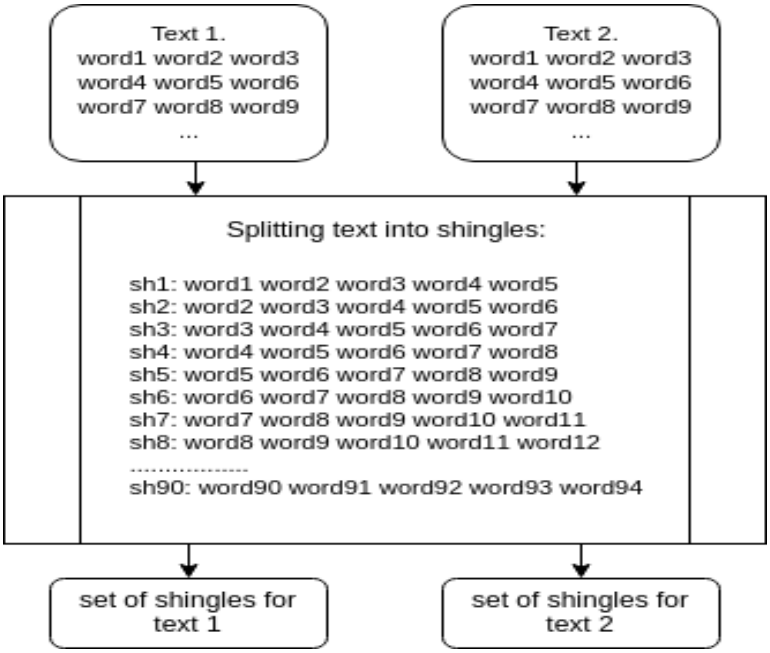


Рис.2. Разделение текста на шинглы.

*Рассчитать контрольную сумму.* После разбивки текста на шинглы нам нужно рассчитать контрольную сумму для каждого шингла. Я использовал только один хэш CRC32. Итак, результат ниже. Но когда у нас много текста, нам нужно вычислить 84 разных хэша и получить минимальное или случайное между ними и сравнить.

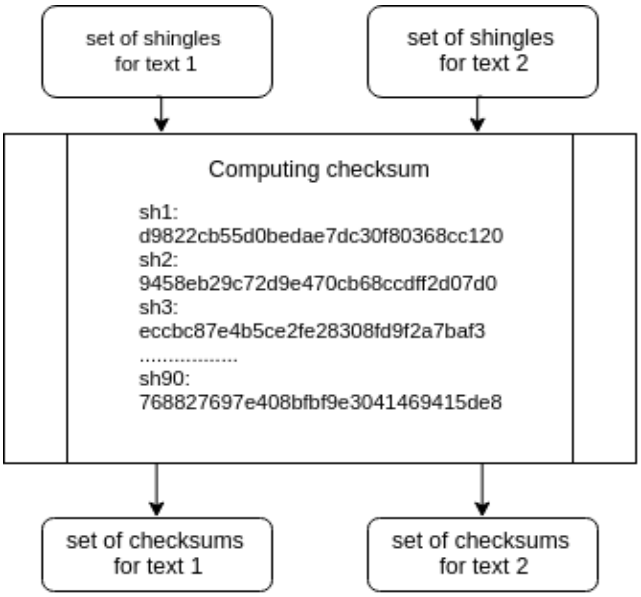


Рис. 3. Расчет контрольных сумм

*Поиск идентичных подпоследовательностей.*

Последний шаг - сравнение хэшей. Сравнение может быть разным. Например, если мы вычислили 84 хэша для каждого шингла, нам нужно найти случайное или минимальное между ними и сравнить попарно между первым и вторым текстом. Нами использован только один хэш. Итак, формулы были: similar_hashes_between_two_sets * 2 / (длина set1 + длина set2) * 100. Эти формулы также называются коэффициентом Jaccard, который дает оценку подобия между двумя наборами. Коэффициент определяется как размер пересечения, количество черепицы, присутствующее в обоих, деленное на размер объединения, количество черепицы, присутствующих в обоих [4]. Если бы у нас была эта оценка подобия для пары документов, мы могли бы определить пороговое значение и сказать, что любые пары документов, которые находятся над данным счетом, являются почти дубликатами [5].

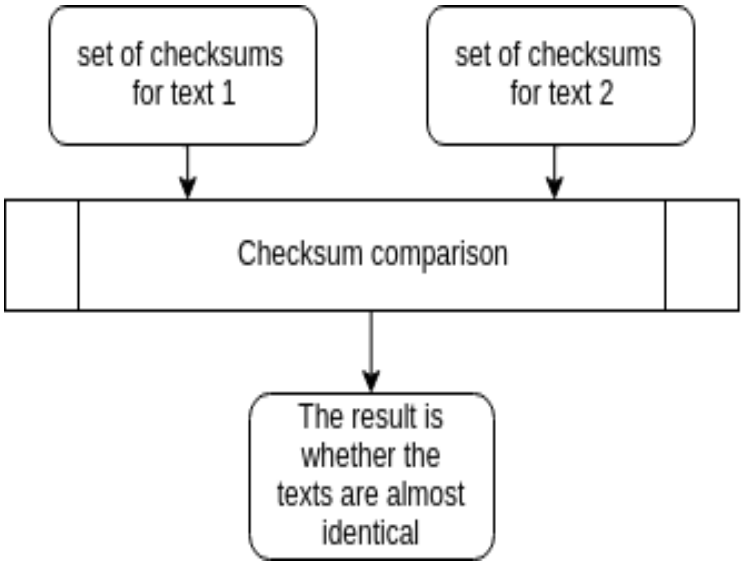


Рис. 4. Сравнение контрольных сумм

***Пример.*** Предположим, что у нас есть два текста для сравнения. Первый из них.

‘BecauseAlmasandZhalgasarrivedatthebusstationbefore noon, I did not see them at the station.’ а второйтекст ‘I did not see them at the station because Almas and Zhalgas arrived at the bus station before noon.’. ПервыйшагпоказаннаДиаграмме 5а. Послеочисткитекстаотстоп-слов и символовостановкинаштекст 1 становится ‘almas zhalgas arrived bus station noon see station’ а текст 2 становится ‘see station almas zhalgas arrived bus station noon’.

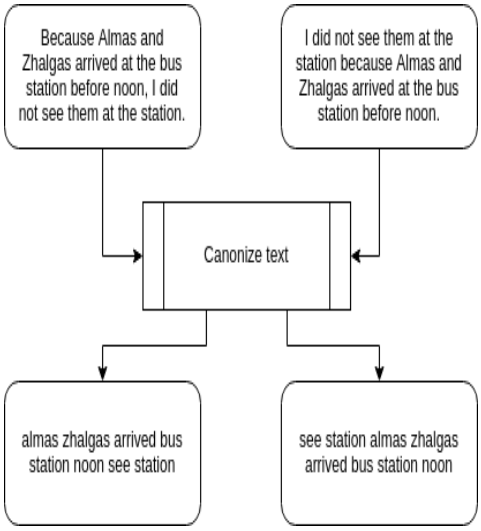


Рис.5a. Step 1

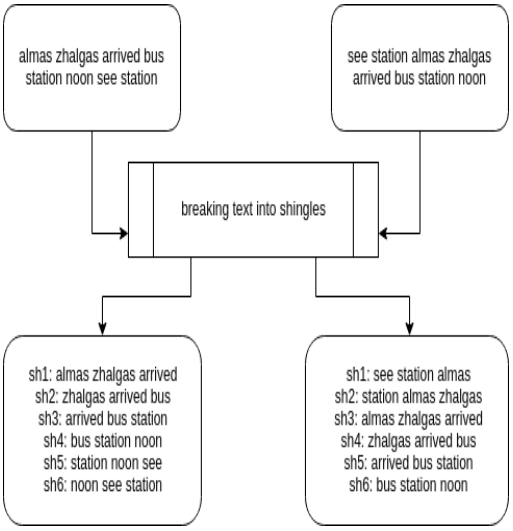


Рис.5b. Step 2

Второй шаг - разделить готовые тексты на шинглы. В моем примере я разделил текст на 3 слова. Итак, у нас будет («количество слов в очищенном тексте» - «длина шингла» + 1 = 8 - 3 + 1) 6 шингла в каждом тексте. Второй шаг показан на Рисунке 5b. После разделение нам нужно вычислить контрольные суммы для каждого шингла. Мой пример это простая реализация алгоритма шингла. Из-за этого я использовал контрольную сумму CRC32. Это показано на рисунке 5c.

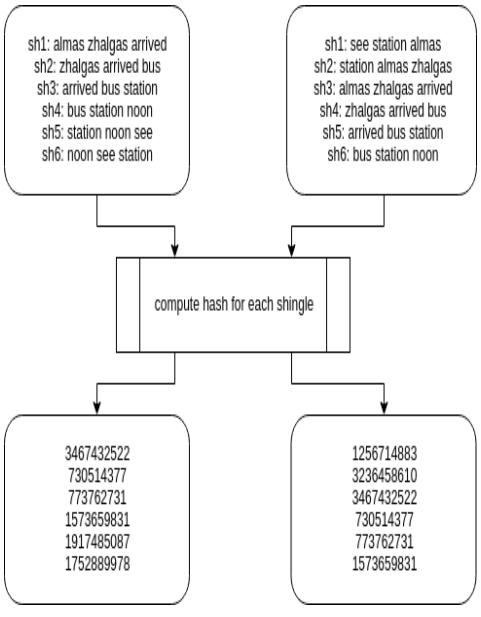


Рис.5c. Step 3

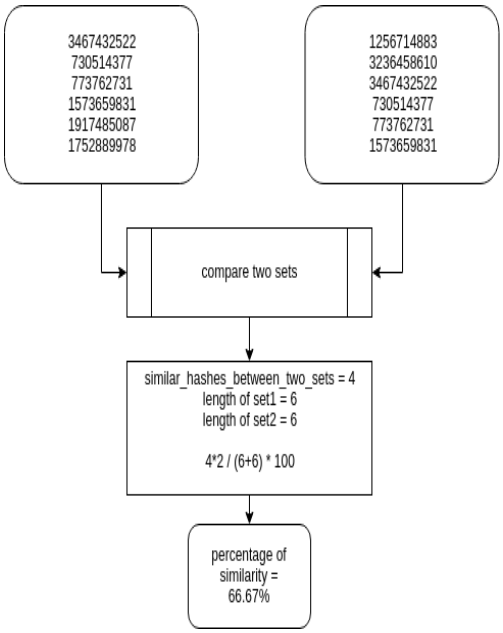


Рис.5d. Step 4

Последний шаг - сравнение наших наборов контрольных сумм и процент вычисления сходства с использованием коэффициента Jaccard. В нашем случае подобная контрольная сумма (пересечение контрольной суммы текста1 и контрольной суммы текста2) будет равна 4. Нам нужно умножить ее на 2. Потому что у нас есть 2 набора. После тогонужно разделить коэффициент пересечения на объединение наборов контрольных сумм. Чтобы найти процент, нужно умножить результат разделения на 100. Таким образом, процент сходства будет 8/12 * 100 = 66,67%.

Таким образом в работе показан один из простых вариантов реализации рассмотренного алгоритма для контрольного примера. Но возможности работы алгоритма можно улучшить, повысив его производительность и точность сравнения текстов.

Сам алгоритм достаточно ресурсоемкий, поэтому будет полезно, если мы создадим механизм кэширования, который будет особенно важен для сравнения наборов файлов, чтобы каждый раз не вычислять контрольные суммы. Например, можно создать базу данных контрольных сумм файлов. После того, как сравним файлы, можно использовать предварительно вычисленные хэши, чтобы найти сходство.

Кроме того, чтобы повысить производительность при обработке больших объемов текста, не обязательно сравнить все полученные контрольные суммы, но только те, которые делятся на 25, или любое целое число от 10 до 40. Как показали экспериментальные тесты, это дает значительное увеличение скорости и не снижает точность. Но это работает только при обработке большого количества текста.

**Список литературы:**

1 Форум технологий Mail.Ru Group: Поиск неточных дубликатов в рунете [интернет ресурс]. – URL: https://www.searchengines.ru/tf_mailru_poisk_netochnyh_dublikatov_v_runete.html

2 NLTK's list of englishstopwords [интернет ресурс]. – URL: https://gist.github.com/sebleier/554280

3 Зеленков Ю.Г, Сегалович И.В. Сравнительный анализ методов определения нечетких дубликатов для Web-документов. – [интернет ресурс]. – URL: http://rcdl2007.pereslavl.ru/papers/paper_65_v1.pdf

4 Shingling - Near Duplicate Detection [интернет ресурс]. - URL: http://phpir.com/shingling-near-duplicate-detection

5 Andrei, Z. Broder. Identifying and Filtering Near-Duplicate Documents[интернет ресурс]. – URL: http://cs.brown.edu/courses/cs253/papers/nearduplicate.pdf