1. **Karp-Rabin & String Matching Algorithm Together.**

* It mainly finds the occurrence of one string into another string. “Finds the occurrence of a pattern in the String”
* In this case, it avoids the comparison between the pattern and the string by characters, it compares them at once. How?
  + It acts upon this question, does a specified pattern exists in a String?
  + It gives each pattern and a word a hash value “it uses a hash functions” – it converts every string to a hash value (e.g. hash(“Global Working”) = 5) – if the hash pattern value is equal to a substring value in the string document then the string may contain the pattern. According to stoimen.com and Stackoverflow, Karp-Rabin algorithm is a good practice for plagiarism detecting because it can deal with multiple patter matching.
* A C# implementation of the algorithm exists. Were the String matching class takes two parameters consisting the pattern(“The possible plagiarized sentence”) and the String Document(“The document I’m searching in”).. and returns an array of the indexes that the algorithm found in the matching patterns.
* Conns:
  + Running time may be a bit high depending on the complexity of the hash
  + Two hashes may equal but the string may not (According to StackOverFlow) which require the substring and the pattern to be compared. Personally, this might not be an issue. Because the matching Hashes are reduced (minimized).

1. **Plagiarism Detection Based on SCAM Algorithm**
   * They count on analysis more than comparison. They extract a sentence from a document. Then, the vocabulary of terms is expanded with synonyms through Wordnet” lexical database for the English language”, trying to cover paraphrasing.
   * A search engine must be used to search for source containing suspicious sentences. But because there is no open source search engine, the uses Apache Lucene java library “full-featured text search engine library written entirely in Java”.
   * The dataset used in their system is restricted to some local documents in text format.
   * For each sentence in each document, several searches are launched, trying to cover all the possible forms of a plagiarized phrase.
   * The dataset is indexed using preprocessed and indexed documents **UNLIKE KARP-RABIN WHICH DOES NOT REQUIRE PREPROCESSING OF DOCUMENTS.** How the index it? They index it in a library in Java called IndexWriter.
   * Going through the SCAM algorithm itself:
     1. SCAM formula detects similarity among documents. This similarity formula returns a high value when the content of test document is either a subset or a superset of the registered document.
     2. **The formula is as following:**

for each retrieved Document

get the terms

for each term

if the term appear in the test Document

get the test term frequency

get the term frequency

if condition EPSILON

calculate S(T, R)

calculate S(R, T)

end if

end if

end for

end for

Personally, The way they describe algorithm sounds possible because the formula sees the sentences as vectors, calculating the similarity between vectors may be a good way for document matching algorithm. But there is no implemented algorithms available. So the way they implement it may be Hard to follow, especially that they use Analysis and search engines. I highly refuse this implementation. I even couldn’t understand it?????

They say:

**We have implemented SCAM (Standard Copy Analysis Mechanism) which is a relative measure to detect overlap by making comparison on a set of words that are common between test document and registered document. Our plagiarism detection system, like many Information Retrieval systems, is evaluated with metrics of precision and recall.**

1. **Local Algorithms for Document Fingerprinting**
   * Talks about fingerprinting algorithms for detecting copying “ quotation, revision, plagiarism, and ﬁle sharing “ in digital content over the web.
   * One of the algorithm they talked about is the Karp-Rabin algorithm mentioned in the first article.
   * They say that The only plagiarism detector algorithm can be used after the Karp-Rabin is the MOSS “Measure Of Software Similarity”.
     1. MOSS is primarily used for detecting plagiarism in programming assignments in computer science and other engineering courses, though several text formats are supported as well.
     2. I found a RUBY implementation of this algorithm. After reading from Quora, the algorithm works in 2 stages. It tokenizes the document, and compares it with another giving document to determine the similarity. I didn’t get deep in the algorithm, but it looks like it’s more for analyzing code. Because one of it’s main tasks is to remove comments line spacing and indentation.
2. **Plagiarism Detection in Java Code**
   * They are mainly focusing “main problem” on detecting the plagiarism that might occur between students submitting Java assignment to a Java Course. It aims to help instructors find if a student cheated from another even when changing attributes names and method names or when changing comments and lines in the code.
   * They talk about multiple ways to detect the plagiarism.
     1. Text-Based Systems: By which they say that it is possible to use online tools such as CopyCatch and WordCHECK that calculate the frequency count of the words in a text, The higher frequency of words and sentences count will indicate higher probability of plagiarism.
     2. Attribute-oriented code-based systems : It’s is more for logic plagiarism detecting. Where it’s specified for detecting code cheating by counting the number of the unique operators and the number of the occurrence of each operator.
     3. They talk about more algorithms that I find out of the topic I’m searching for.
   * They also talk about an algorithm named Levenshtein Distance. This was one of the first algorithms I read about. It calculate the similarity percentage between 2 sentences. After implementing it, I noticed that “hello” and “hllo” have a high percentage similarity. After researching, I found that this algorithm is used for smaller and short text and words. It’s MAIN application is a spelling checker. Which is far away of what we’re looking for!