UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

COORDENAÇÃO DE ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

TARCISIO SPAK

IMPLEMENTAÇÃO DE UM ALGORITMO PARA AUXÍLIO NA DETECÇÃO DE PLÁGIO EM TRABALHOS ACADÊMICOS

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA

2012

TARCISIO SPAK

IMPLEMENTAÇÃO DE UM ALGORITMO PARA AUXÍLIO NA DETECÇÃO DE PLÁGIO EM TRABALHOS ACADÊMICOS

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas, da Universidade Tecnológica Federal do Paraná.
Orientador: Prof. Dr. Gleifer Vaz Alves
Co-Orientador: Prof. Ademir Mazer Jr.

PONTA GROSSA

2012

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa

Diretoria de Graduação e Educação Profissional

TERMO DE APROVAÇÃO

IMPLEMENTAÇÃO DE UM ALGORITMO PARA AUXÍLIO NA DETECÇÃO DE PLÁGIO EM TRABALHOS ACADÊMICOS

por

TARCISIO SPAK

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 31 de Maio de 2012 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

_____
Gleifer Vaz Alves
Prof. Orientador(a)

_____
Ademir Mazer Jr.
Membro titular

_____
Saulo Jorge Beltrão Queiroz
Membro titular

_____
Helyane Bronoski Borges
Responsável pelos Trabalhos
de Conclusão de Curso

_____
Simone Almeida
Coordenador do Curso
UTFPR - Campus Ponta Grossa

Dedico este trabalho à minha família, amigos e colegas e professores da UTFPR.

AGRADECIMENTOS

Agradeço a toda a minha família e amigos pelo apoio e motivação prestados durante todo desenvolvimento deste trabalho, aos professores da UTFPR em especial ao professor Ademir Mazer Jr. e Gleifer Vaz pela dedicação na orientação e acompanhamento, aos quais foram de grande importância para que este trabalho fosse concluído dentro do prazo determinado.

RESUMO

SPAK, Tarcísio. **Implementação de um algoritmo para Auxílio na Detecção de Plágio em Trabalhos Acadêmicos**. 2012. 50f. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) – Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2012.

A prática do plágio é algo bastante comum nas universidades do mundo todo. Trabalhos de colegas e produções textuais disponíveis na Internet são comumente copiados por terceiros e atribuídos como de sua própria autoria sem referenciá-los devidamente. As diferentes formas utilizadas por plagiários tornam a tarefa da detecção mais difícil. Desta forma, o objetivo deste trabalho é implementar um algoritmo de detecção de plágio utilizando o conceito n-grama, o qual foi escolhido através de um estudo sobre métodos de detecção existentes. Além disso, a ferramenta Lucene é utilizada no trabalho para indexação e busca de documentos.

Palavras-chave: Plágio, Métodos de Detecção de Plágio, Lucene, N-grama.

ABSTRACT

SPAK, Tarcísio. **Implementation of algorithm to facilitate the detection of Plagiarism in Academic Papers.** 2012. 50p. Trabalho de Conclusão de Curso (Tecnologia em Análise e Desenvolvimento de Sistemas) – Federal Technology University - Paraná. Ponta Grossa, 2012.

The practice of plagiarism is very usual in universities around the world. Papers from colleagues and other textual productions available on the Internet are commonly copied by third parties and assigned as of their own without properly referencing. The amount of forms used for plagiarists turns the task of detection a very hard work. Thus, the major goal of our work is the creation of an algorithm for plagiarism detection by means of the n-gram concept, where such concept has been chosen through a survey of existing detection methods. Moreover, the Lucene's tool has been used for indexing and searching documents.

Keywords: Plagiarism, Methods of Detecting Plagiarism, Lucene, N-gram.

LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS

API	*Application Programming Interface* (Interface de Programação de Aplicativos)
DNA	*DeoxyriboNucleic Acid* (Ácido DesoxirriboNucleico)
HTTP	*Hyper Text Transfer Protocol* (Protocolo de Transferência de Hipertexto)
PDF	*Portable Document Format* (Formato de Documento Portável)
REST	*Representational State Transfer* (Transferência de Estado Representacional)
SOA	*Service-Oriented Arquitecture* (Arquitetura Orientada a Serviços)
URI	*Uniform Resource Identifier* (Identificador Uniforme de Recursos)
XML	*eXtensible Markup Language* (Linguagem de Marcação Estendida)

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Nos dias atuais, onde o volume de informações disponíveis é grande e o acesso a tais informações está cada vez mais facilitado, seja através da Internet ou de qualquer outro meio de comunicação, surge o problema da confiabilidade no que diz respeito à autoria de trabalhos, artigos e outros tipos de produções textuais que se tem produzido.

Nas universidades, onde o número de trabalhos textuais é exigido mais frequentemente, esta preocupação é ainda maior. Pois tais trabalhos são meios de avaliar o conhecimento de cada aluno. Sendo assim, os mesmos devem ser confiáveis a ponto de dar o devido crédito de quem às produziu. Porém, nem sempre isto acontece, pois textos e ideias são comumente copiados, por acadêmicos nas universidades, atribuindo-se como de própria autoria.

1.1 PLÁGIO

Nas últimas décadas, o termo plágio tem ganhado ênfase em várias discussões, dissertações e artigos devido ao fato da grande preocupação que se tem atualmente sobre este ato ilegal. Nas instituições de ensino, como universidades, a situação é ainda mais preocupante, pois, além de ser uma prática ilegal é também imoral o que compromete a educação do universitário e a integridade da instituição.

O ato de plagiar qualquer tipo de obra que caracterize como propriedade intelectual já vem de tempos antigos. Atualmente com a Internet e a grande facilidade de acesso às informações que ela proporciona, esse crime tem sido cometido mais frequentemente (GALVÃO; OLIVEIRA, 2011).

Segundo ROMANCINI (2007), plágio acadêmico caracteriza-se como:

> “uma falsa atribuição de autoria, uma apropriação indevida de trabalho de um autor por outro indivíduo (o plagiário). Em outras palavras, trata-se da cópia de ideias ou conteúdos de trabalhos de outra pessoa, que são utilizados como se fossem daquele que finge ser o autor legítimo dos mesmos.”

Em seus estudos, (ALZAHRANI; SALIM; AJITH, 2011) classificam o plágio acadêmico em dois tipos: o *literal* e o *inteligente*.

1.1.1 Plágio Literal

O plágio literal é caracterizado pela cópia idêntica do texto ou de parte dele, a modificação da estrutura e ordem da frase e ou a modificação através da inserção, exclusão ou substituição de palavras. Por ser facilmente realizado é mais comumente praticado pelos alunos.

1.1.2 Plágio Inteligente

No plágio inteligente o indivíduo plagiário tenta esconder de várias formas o trabalho realizado por outro autor, apresentando-o como de sua própria autoria. Nesta forma de plágio são utilizadas estratégias como: manipulação de texto, tradução e adoção de ideia(s) (ALZAHRANI; SALIM; AJITH, 2011).

Na manipulação de texto é utilizada a paráfrase, que consiste em modificar algumas palavras pelos seus sinônimos ou antônimos mantendo o mesmo significado original. Também o resumo através da redução, combinação e restruturação de sentenças, assim como a generalização e especificação de conceitos são considerados plágio caso não sejam devidamente referenciados (ALZAHRANI; SALIM; AJITH, 2011).

A tradução de textos é outra forma de plágio inteligente, onde o todo ou uma parte de texto é transcrito em outra língua omitindo o verdadeiro autor. As traduções são realizadas manualmente por pessoas que dominam os idiomas utilizados ou então por ferramentas automatizadas como, por exemplo, o tradutor da Google (ALZAHRANI; SALIM; AJITH, 2011).

Na adoção de ideia(s), a forma mais crítica de cometer o plágio, o individuo plagiário utiliza-se de ideias, contribuições, resultados e conclusões de outros autores, apresentando-as como suas (ALZAHRANI; SALIM; AJITH, 2011).

1.2 OBJETIVO GERAL

O objetivo deste trabalho é auxiliar ao docente no combate ao plágio literal em trabalhos acadêmicos, através da implementação de uma técnica de detecção de plágio, utilizando a mesma estrutura do framework desenvolvido por JACHINSKI e GELINSKI (2011). Este Framework foi desenvolvido em um trabalho relacionado e faz uso de uma estrutura de serviços web. Desta forma, a ideia é que a ferramenta seja capaz de identificar trechos de texto idênticos em documentos distintos apresentando como resultado, quais documentos possuem textos possivelmente copiados.

1.3 OBJETIVOS ESPECÍFICOS

- Pesquisar os diferentes métodos de detecção de plágio existentes na literatura;
- Manter um índice local de documentos indexados através da biblioteca Apache Lucene;
- Implementar uma técnica de detecção de plágio baseada na abordagem n-grama, mantendo a mesma estrutura de serviços web desenvolvidos por JACHINSKI e GELINSKI (2011).

1.4 ORGANIZAÇÃO DO TRABALHO

O trabalho aqui desenvolvido está estruturado da seguinte forma: no capítulo 2 serão abordados os métodos de detecção de plágio, técnicas de processamento de texto e alguns trabalhos que possuem o tema relacionado ao plágio e sua detecção. No capítulo 3 será explicado como foi implementada a técnica, assim como as ferramentas utilizadas, os testes realizados e os problemas encontrados. E por último a conclusão e os possíveis trabalhos futúros.

2 DETECÇÃO DE PLÁGIO: MÉTODOS E TÉCNICAS

No decorrer deste capítulo serão descritos os métodos mais comuns de detecção de plágio, técnicas de processamento de texto que de alguma forma auxiliam na análise e detecção e trabalhos acadêmicos que abordaram assuntos relacionados a plágio, análise de similaridades e ferramentas de detecção.

2.1 MÉTODOS DE DETECÇÃO DE PLÁGIO

Segundo ABREU (2011), a dificuldade em identificar os algoritmos utilizados nas principais ferramentas de detecção de plágio, está no fato de que os mesmos são tratados como segredos industriais. Porém, no campo da literatura é possível encontrar diferentes técnicas e métodos que auxiliam no processo de levantar possíveis similaridades entre textos.

Nesta seção, serão abordados alguns métodos de detecção de plágio, baseado no trabalho de (ALZAHRANI; SALIM; AJITH, 2011) seguindo a sua classificação. A escolha deste trabalho como referência se deve ao fato do mesmo realizar uma abordagem mais genérica sobre os diferentes tipos de plágio existentes e em cada caso o método utilizado para a sua detecção.

Segundo o estudo, os métodos para comparação e manipulação de texto para avaliar possíveis fraudes classificam-se e baseiam-se em oito tipos: caracteres, vetores, sintaxes, estruturas semânticas, *Fuzzy*, estrutura da escrita, estilometria e Cross linguais. Nas próximas subseções serão abordadas cada uma destas técnicas.

2.1.1 Métodos Baseados em Caracteres

O método baseado em caracteres utiliza técnicas e algoritmos que manipulam recursos lexicais de caracteres ou palavras e sintaxes de sentenças para comparar a consulta de um documento *dq* com um documento candidato $dx \epsilon DX$.

Uma das principais técnicas utilizadas para tal é a abordagem n-grama. Um n-grama consiste em "uma sequência de *n* letras ou palavras, onde *n* geralmente é 1, 2 ou 3, respectivamente monograma, bigrama e trigrama" (ALMEIDA; FERREIRA, 2004). Neste caso, utilizando a abordagem de caracteres 3-grama para a palavra "plágio" resultaria em: "plá", "lág", "ági" e "gio".

Nesta técnica são utilizados algoritmos que consideram apenas sentenças iguais ou então aproximadas.

Para a combinação de duas sentenças serem iguais, devem ter necessariamente o mesmo número de caracteres e em mesma ordem. Por exemplo, a abordagem de caracteres 8-grama para *x="aaabbbcc"* é igual a *z="aaabbbcc"* mas diferente de *y="aaabbbcd".*

Algoritmos que consideram sentenças aproximadas utilizam métricas para definir o grau de similaridade. Por exemplo, para caracteres 9-grama *x="aaabbbccc"* e *y="aaabbbccd"* são altamente semelhantes, exceto por uma letra.

Para as sentenças aproximadas são utilizados algoritmos de edição de distância como, por exemplo, o Levenshtein Distance, que consiste em realizar a menor quantidade de operações para transformar uma sentença em outra. As operações possíveis são: inserção, exclusão e substituição, custando um peso para cada operação. Por exemplo, para obtermos a distância *D(x, y)* para *x="aabbccdd"* e *y="aabbcdef"*, é necessário substituir os caracteres "def" da sentença *y* por "cdd" da sentença *x*, de forma a obtermos *D(x, y)* = 3 (ALZAHRANI; SALIM; AJITH, 2011).

Também destaca-se, uma das técnicas baseadas em carácteres, a Máxima Subsequência Comum (LCS - Longest Common Subsequence), que consiste em encontrar a máxima similaridade entre duas sentenças. Por exemplo, tendo S1 = "jhlbcdehk" e S2 = "abcdefg" a LSC entre S1 e S2 resultaria em S3 = "bcde". Desta forma, a técnica compara cada um dos caracteres de S1 com cada caractere contido na sentença S2 e isto pode tornar a tarefa um pouco lenta quando se trata de sentenças longas (AMJAD; MOHAMED, 2009).

2.1.2 Métodos Baseados em Vetores

Utiliza o conceito de tratar recursos lexicais e sintaxes em forma de um vetor, armazenando cada termo presente em uma sentença. Neste caso, para a

abordagem n-grama usando palavras, cada n-grama é representado como um vetor de n termos ou palavras. Frases ou sentenças podem ser tratadas tanto como vetores de palavras ou de caracteres (ALZAHRANI; SALIM; AJITH, 2011).

A similaridade pode ser calculada através de métricas de vetores de similaridade. Como por exemplo, a métrica Jaccard que separa a quantidade de termos comuns contra o total de termos. Esta medida é utilizada para verificar se os vetores são iguais.

Outra métrica, também utilizada para detecção de plágio, denominada coeficiente do cosseno, utiliza fórmulas para encontrar o ângulo cosseno entres dois vetores para encontrar similaridade entre os mesmos.

2.1.3 Métodos Baseados em Sintaxes

Este método se baseia em recursos sintáticos para verificação de similaridade entre textos e detecção de plágio. Desta forma, tem-se por base a intuição de que cópias similares de documentos terão estruturas sintáticas similares.

Para obter o cálculo de similaridade entre textos é utilizada recurso de *Part-of-Spech* (POS), que consiste em uma marcação ou classificação gramatical, como por exemplo, os substantivos, adjetivos, verbos, advérbios, etc. Desta forma documentos similares que contém partes textuais iguais ou próximas de iguais de outros documentos, podem também conter a mesma estrutura sintática (ALZAHRANI; SALIM; AJITH, 2011).

2.1.4 Métodos Baseados em Estruturas Semânticas

Este método está focado em identificar similaridade entre textos que contenham diferentes estruturas e ordem de palavras assim como as diferentes palavras que possuem o mesmo significado em uma linguagem. Ou seja, duas frases podem ter o mesmo sentido, porém, são diferentes em sua estrutura. Como por exemplo, pode-se utilizar a voz ativa ou a passiva ou então ao invés de escrever determinada palavra pode-se utilizar o seu sinônimo ou antônimo, mantendo o mesmo significado da frase (ALZAHRANI; SALIM; AJITH, 2011).

Por razão da dificuldade que se tem em encontrar similaridades semânticas entre texto, são poucos os sistemas de detecção de plágio que utilizam este método. Duas sentenças podem ser semanticamente similares através da similaridade entre suas palavras ou a ordem em que se encontram.

Uma ferramenta bastante útil para esta abordagem é WordNet que é uma base de conhecimento linguístico contendo o significado lexical e semântico de palavras e expressões. A versão em português conta, atualmente, com 19 mil expressões em sua base (WORDNET, 2012).

2.1.5 Métodos Baseados em Aproximação

Esta técnica utiliza a filosofia de que a análise de fragmentos textuais como as sentenças podem ser aproximadas ou diferentes. Desta forma, implementa-se uma escala de similaridade que varia de zero (diferente) a um (idêntico).

Na detecção de plágio este conceito pode considerar que cada palavra em um documento é associada a um conjunto fuzzy de palavras que possuem o mesmo significado e há um grau de similaridade entre as palavras do documento e o conjunto.

Na teoria, este método é considerado eficaz para detectar plágio, pois, consegue detectar frases com estruturas diferentes e, no entanto com o mesmo significado, porém a dificuldade maior está em modelar um conjunto *fuzzy* e o grau de similaridades entre as palavras (ALZAHRANI; SALIM; AJITH, 2011).

2.1.6 Métodos Baseados em Estruturas Textuais

Enquanto que os métodos citados anteriormente se concentram em encontrar similaridades textuais através de recursos lexicais, sintáticos e semânticos, esta técnica está focada em encontrar similaridades baseando-se no contexto em que as palavras são usadas no texto, ou seja, nas seções e parágrafos (ALZAHRANI; SALIM; AJITH, 2011).

Umas das formas de se implementar tal método é utilizando o modelo de auto organização de mapa multicamada (ML-SOM), em inglês, *Multi-Layer Self-Organization Map,* que trabalha com estrutura de árvores de dados.

2.1.7 Métodos Baseados em Estilometria

Esta técnica tem como característica identificar padrões de estilo de escrita do autor utilizando fórmulas que caracterizem cada texto. As fórmulas podem ser classificadas baseando-se especificamente na escrita, medindo o seu vocabulário e complexidade do estilo de escrita do autor ou então, na leitura, utilizando fórmulas que classifiquem o nível necessário para compreender um texto (ALZAHRANI; SALIM; AJITH, 2011).

2.1.8 Métodos Cross-Linguais

A última técnica, aqui apresentada, é utilizada para detectar plágio em documentos que possuem idiomas diferentes. Desta forma, sentenças de um documento suspeito são medidas com sentenças de um outro documento candidato, de forma a obter similaridade entre os mesmos baseando-se em recursos de cada idioma (ALZAHRANI; SALIM; AJITH, 2011).

Entre os métodos utilizados para este fim estão o uso n-grama para texto sintaticamente semelhantes, como as línguas europeias, baseado em dicionários e sua relações semânticas, métodos relacionados em estatística e entre outros.

2.1.9 O Método Escolhido

Das técnicas descritas nas sessões anteriores, escolheu-se implementar no presente trabalho, o método baseado em vetores em conjunto com a abordagem n-grama para a geração das consultas. Optou-se por esta técnica pelo fato ser eficiente no que se propõem e se encaixar perfeitamente no escopo do trabalho, que

é a implementação de um método que auxilie na detecção de plágio onde o intuito é identificar frases idênticas em diferentes documentos.

Como será utilizada a biblioteca Lucene (mais bem detalhada posteriormente) a principal preocupação do algoritmo concentra-se na geração das consultas e na organização dos resultados. Desta forma, a técnica n-grama é a melhor das opções, citadas anteriormente, para este propósito.

2.2 TÉCNICAS DE PROCESSAMENTO DE TEXTO

Para a tarefa de detecção de plágio algumas palavras e caracteres são considerados irrelevantes por serem comumente usadas e não representarem, por si só, significado algum. Assim também como as várias derivações resultantes de uma mesma palavra que possuem o mesmo significado, também devem ser detectadas. Duas técnicas bastante conhecidas e utilizadas para o pré-processamento de texto são as *stop words* e *stemming.*

Stop words é uma lista de palavras que são consideradas irrelevantes para uma análise de comparação de texto e não caracterizam-se como plágio, pois são palavras que são consideradas artigos, pronomes, preposições e entre outros. Cada idioma possui sua própria lista de *stop words* (ABREU, 2011).

Outra técnica de pré-processamento é *stemming,* que consiste em transformar cada palavra do texto analisado em seu radical ou raiz. Muitas vezes uma mesma palavra possui várias derivações e que mantém o mesmo significado. Desta forma, o processo de *stemming* visa identificar tais palavras considerando-as iguais. Assim como as *stop words*, esta técnica também está fortemente atrelada ao idioma utilizado *(ABREU, 2011).*

2.3 TRABALHOS RELACIONADOS

Nesta seção é realizado um breve resumo de trabalhos acadêmicos, encontrados na literatura, que abordam assuntos relacionados a sistemas de detecção de plágio, o plágio em si e a suas classificações. Neste tópico, optou-se

por pesquisar trabalhos que tivessem alguma relação com o plágio ou ferramentas que auxiliem na detecção, pois além de implementar o algoritmo aqui proposto, neste trabalho também o foco é a pesquisa de técnicas existentes de detecção de plágio.

2.3.1 Ferramenta de Detecção de Plágio em Ambiente Virtual

Desenvolvido por (SANTOS, 2010), o sistema batizado de Araponga e integrado ao ambiente virtual TelEduc é uma melhoria do software de detecção de plágio Sherlock, que por sua vez, tem como características, encontrar similaridades em textos através de assinaturas digitais, onde para cada certo número de palavras é gerado uma assinatura e ao final deste processo é feito a comparação, gerando a o percentual de similaridades entres os textos (PIKE, 2012).

O funcionamento do Software Araponga é realizado em duas etapas: o processamento e a comparação.

O processamento consiste em separar o texto em frases e de frases em palavras. Na separação das frases é usado o ponto final para a delimitação e então são removidos os caracteres inválidos, tais como: acentos, diferenciação de letras minúsculas e maiúsculas, linhas em branco e conectivos. O próximo passo é identificar cada frase por um vetor. E então, cada elemento de um vetor é comparado com todos os outros elementos de outros vetores. Quando houver coincidência incrementa-se uma variável que mostrará a quantidade de palavras repetidas nas duas frases. Esta etapa de processamento é executada em todos os textos dos alunos da turma (SANTOS, 2010).

Na próxima etapa é realizada a comparação entre as matrizes. Ou seja, a matriz de palavras do texto original é comparada com a matriz do texto suspeito. E então após retornar o grau de similaridade entre os dois textos, verifica-se a similaridade é maior que a diferença obtendo assim a porcentagem de plágio.

O sistema também permite a pesquisa de qualquer uma das frases, original ou suspeita redirecionando, em outra janela, para o site da Google.

2.3.2 Levenshtein Distance para Cálculo de Similaridade

O estudo realizado por (OLIVEIRA, 2010) aborda vários conceitos e técnicas de comparação entre textos. No entanto, enfatiza uma abordagem mais aprofundada sobre o algoritmo Levenshtein Distance.

Levenshtein Distance foi criado em 1965, pelo cientista russo Vladimir Levenshtein. É utilizado na comparação de dialetos, correção ortográfica, reconhecimento da fala, análise de DNA e entre outros.

O algoritmo tem como principal característica a análise de similaridade entre duas *strings* tendo como base a quantidade de operações necessárias para transformar uma *string* em outra, sendo as operações a inserção, exclusão e substituição. A distância zero significa que as *strings* são iguais. Através do tamanho de cada string é criada uma matriz onde são armazenados os custos de cada operação e então finalmente a distância é obtida pela última posição da matriz (OLIVEIRA, 2010).

Para a aplicação do algoritmo acima descrito é realizado antes um pré-processamento do texto, como a remoção de *stopwords* e a aplicação da técnica de *stemming*.

2.3.3 Arquitetura para Detecção de Plágio Multi-algoritmo

O trabalho desenvolvido por (ABREU, 2011) aborda vários conceitos e técnicas relacionadas ao plágio e a sua detecção e propõem uma arquitetura combinando a utilização de alguns destes conceitos.

A proposta de arquitetura divide o trabalho em cinco fases: a extração de consultas, recuperação de documentos, análise de documentos, análise de plágio e geração de resultados.

O protótipo batizado de TEXPLAN, também realiza a etapa de pré-processamento. Além disso, utiliza-se de ferramentas, tais como o Lucene que será detalhada posteriormente, para utilizar sua lista de *stop words*, o algoritmo Levenshtein Distance para a análise de similaridade e *WordNet* para a utilização de sinônimos (ABREU, 2011).

2.3.4 Padrões de Plágios e Métodos de Detecção

No estudo realizado por (ALZAHRANI; SALIM; AJITH, 2011) é feita uma classificação, através de pesquisa com docentes e discentes de várias universidades, sobre o plágio, os diferentes padrões utilizados por plagiários e os métodos mais comuns de detecção para cada tipo.

Em (ALZAHRANI; SALIM; AJITH, 2011) os autores classificam o plágio o plágio em literal e inteligente.

Quanto aos métodos de detecção de plágio, são classificados em oito tipos: baseados em caracteres, vetores, sintaxes, estruturas semânticas, Fuzzy, estrutura da escrita, estilometria e *Cross* linguais (ALZAHRANI; SALIM; AJITH, 2011).

2.3.5 Avaliação de Ferramentas para Identificação de Plágio

Nesta monografia, desenvolvida por (GALVÃO; OLIVEIRA, 2011), é realizado um estudo aprofundado sobre o plágio e seus conceitos, citando a sua origem e seu histórico assim como a legislação relacionada a este assunto no Brasil e mundialmente.

O principal foco do trabalho é realizar um levantamento sobre as ferramentas de detecção de plágio disponíveis no mercado, levando em consideração aspectos como, acessibilidade, quais formatos de arquivo são aceitos, integração a bases de dados locais e de repositórios de trabalhos acadêmicos e forma de aquisição, dando ênfase para softwares livres ou de baixo custo.

Segundo pesquisas realizadas pelo estudo, a maioria das Universidades entrevistadas não possui verba o suficiente para obter ferramentas automatizadas que auxiliem aos professores na tarefa de detectar plágio. Por isso, em muitas Instituições ainda são utilizados meios convencionais, como a busca manual e realização de bancas para tal problema (GALVÃO; OLIVEIRA, 2011).

No trabalho de GALVÃO e OLIVEIRA (2011) também é feita uma análise comparativa de ferramentas de detecção de plágio existentes no mercado, avaliando o funcionamento de cada sistema.

2.3.6 Ferramenta para Indexação e Verificação de Plágio

O trabalho de (JACHINSKI; GELINSKI, 2011) tem o propósito de implementar uma ferramenta para detecção de plágio baseando-se em Web Service. Para tal, são utilizados conceitos como SOA (Arquitetura Orientada a Serviços) e ferramentas como Restful para implementação de Web Service e Apache Lucene para indexação de documentos textuais e buscas de similaridades entre textos.

Em geral, o software está mais focado no funcionamento e integração entre as ferramentas do que na própria descoberta de plágio. Seu funcionamento resume-se em duas etapas, a indexação de arquivo e a busca de similaridades.

Na indexação é permitido ao usuário enviar o documento a ser indexado para a posterior análise. Como resultado, o serviço retorna uma página em XML mostrando o resultado da indexação.

Na fase da busca é possível o usuário enviar o documento que será analisado. Desta forma, é realizado um pré-processamento do texto removendo caracteres e excesso de espaços, então o texto é fracionado em frases que serão utilizadas para a consulta.

Como resultado da análise são apresentados os documentos suspeitos de plágio com o seus respectivos: id, total, encontrados, percentual e nome. Onde, total é a quantidade de frases usadas como pesquisa, encontrados é o total de frases que retornaram similaridades, o percentual é a quantidade de similaridades em relação ao total (JACHINSKI;GELINSKI, 2011).

3 DESENVOLVIMENTO DA FERRAMENTA DE DETECÇÃO DE PLÁGIO

No decorrer deste capitulo serão abordados os problemas ao qual este trabalho se propõe a resolver, as técnicas e ferramentas utilizadas e a arquitetura do projeto em si.

Primeiramente será feito uma breve abordagem dos conceitos e ferramentas utilizados para o desenvolvimento deste trabalho e da estrutura do framework desenvolvido por JACHINSKI e GELINSKI (2011), para depois entrar em detalhes de como foi realizado o trabalho aqui proposto.

3.1 CONCEITOS E FERRAMENTAS UTILIZADAS

Nesta seção serão brevemente detalhados os conceitos e ferramentas utilizados para o desenvolvimento deste trabalho, proposto inicialmente.

3.1.1 SOA

SOA (Service-Oriented Architeture), pode ser traduzida como Arquitetura Orientada a Serviços, tem como principais características a de garantir interoperabilidade, escalabilidade e baixo acoplamento entre sistemas e serviços.

Como qualquer outro conceito ou tecnologia existente, SOA é uma forma de abstrair. Ao contrário de outras tecnologias que são simplesmente compradas e implantadas, SOA deve ser primeiramente entendida (JOSUTTIS, 2007).

Segundo JOSUTTIS (2007), SOA é uma abordagem que ajuda os sistemas a manter a escalabilidade e flexibilidade enquanto crescem. Três elementos principais compõem a abordagem: os serviços, a interoperabilidade e o baixo acoplamento.

O serviço é uma funcionalidade auto-suficiente que pode ser simples, como por exemplo, o cadastro de um cliente ou então, mais complexo como realizar um

processamento para outro cliente. Um mesmo serviço pode ser responsável por uma ou mais tarefas.

A interoperabilidade é um conceito que garante que sistemas desenvolvidos por tecnologias e plataformas diferentes possam utilizar-se dos mesmos serviços. A ESB (Enteprise Service Bus) é uma arquitetura bastante utilizada para garantir alta interoperabilidade entre os sistemas distribuídos e os serviços.

O baixo acoplamento é o conceito de minimizar as dependências do sistema, de modo que minimize os efeitos de modificações e falhas. Contudo, o acoplamento flexível torna-se complexo, pois, são mais difíceis de desenvolver, manter e depurar (JOSUTTIS, 2007).

3.1.2 REST

O conceito REST (Representational State Transfer) ou Transferência de Estado Representacional é um estilo de arquitetura que provê um conjunto de normas e conceitos que visam facilitar a abstração e criação de Web Services (COSTELLO, 2012).

Segundo FIELDING e TAYLOR:

> "REST é um conjunto coordenado de restrições arquiteturais que visa minimizar a latência e comunicação na rede enquanto que ao mesmo tempo maximiza a independência e escalabilidade na implementação de componentes." (FIELDING; TAYLOR, 2000 – Tradução Livre).

Seguindo estes conceitos, pode se afirmar que a própria Internet (*World Wide Web*) é uma instância dos padrões estabelecidos no estilo REST.

As principais características de aplicações RESTful (denominação para sistemas que utilizam o conceito REST) é utilizar de uma forma mais eficiente os recursos e tecnologias existentes, tais como: HTTP e URIs. Para tal, são estabelecidas algumas restrições como: a aplicação deve ser desenvolvida em uma plataforma Cliente-Servidor, cada recurso ou serviço deve ser independente (sem estado), deve suportar a um sistema de armazenamento em cache provido pela infraestrutura da rede, endereçamento único e válido para cada recurso e deve suportar escalabilidade (SANDOVAL, 2009).

Um dos frameworks existentes para auxiliar na implementação de um sistema RESTful, o JERSEY ou simplesmente JAX-RS, foi desenvolvido para

auxiliar na utilização dos conceitos REST para aplicações desenvolvidas em linguagem JAVA (SANDOVAL, 2009).

3.1.3 Apache Lucene

Quando se trabalha com sistemas de informações, onde o crescimento do volume de dados é inevitável e imprevisível, é necessário dispor-se de mecanismos e ferramentas que tornem a tarefa de manipulação destas informações muito mais ágeis, eficientes e que demandem o mínimo de processamento possível. Lucene, A API de indexação e busca de dados, foi desenvolvida para este propósito.

Desenvolvida por Doug Cutting, a biblioteca Lucene é uma ferramenta de alto desempenho, que possui eficiente método de indexação e um poderoso motor de busca. Escrita em Java e posteriormente unida com a organização *Apache Software Foundation*, a biblioteca é utilizada em aplicações de muitas Organizações que necessitam prover mais agilidade e eficiência na recuperação das informações (MCCANDLESS; HATCHER; GOSPODNETIC, 2010).

O funcionamento da biblioteca consiste primeiramente na indexação do texto e depois na própria pesquisa. Na indexação os dados são processados, mantendo-se o texto original e organizados em uma estrutura inter-relacionada, facilitando a busca. Para tal, Lucene mantem um índice de termos (palavras) invertidos, o que torna muito mais eficiente a busca por similaridade quando é executada a consulta.

Segundo MCCANDLESS (2010) *et al*, as seguintes classes são necessárias para realizar a indexação:

- *IndexWriter* – Componente central do processo de indexação, é a responsável por criar um índice ou abrir algum já existente e adicionar, atualizar e remover documentos no mesmo;
- *Directory* – Diretório onde é realizado a indexação do texto. O diretório para este armazenamento pode ser na máquina local, através da subclasse *FSDirectory* ou então na memória através da subclasse *RAMDirectory*;
- *Analyzer* – Classe abstrata, responsável por extrair e carregar *tokens* (palavras) do texto que serão indexados, eliminando o restante. Caso o texto não esteja no formato adequado, é necessário realizar antes a conversão.

Uma de suas principais funcionalidades é permitir que palavras, irrelevantes para a consulta, sejam ignoradas, este processo chama-se *stop words*;

- *Document* – Representa a coleção de atributos de um determinado documento, como por exemplo: autor, título, assunto e data de modificação. Para cada arquivo de texto a ser indexado, é criado uma nova instância da classe *Document* e atribuídos seus respectivos campos;
- *Field* – Classe que contém um ou vários atributos de um documento. Cada atributo corresponde uma parte dos dados, a qual tem a sua devida importância para a recuperação e consulta durante a execução da pesquisa. São quatro os diferentes tipos de atributos oferecidos por esta classe: *Keyword*, dados não analisados, porém indexados. Por exemplo: datas, nomes pessoais, URL, etc; *UnIndexed*, dados não indexados e não analisados, mas são armazenados para posteriormente serem mostrados em resultados. Exemplo: URL e chave primária; *UnStored*, dados são analisados e indexados, porém não armazenados, como por exemplo o corpo de um página web; e *Text*, é analisado, indexado.

Para a busca de similaridades é necessário a utilização das seguintes classes (MCCANDLESS; HATCHER; GOSPODNETIC, 2010):

- *IndexSearcher* – é a classe responsável por abrir o índice e dar o acesso a leitura dos dados. Como parâmetro o método construtor da classe recebe o diretório onde se encontra o índice;
- *Term* – é a unidade básica para a pesquisa, consiste em um par de elementos *String*: o nome do campo (*field*) e o valor do mesmo. É similar ao *Field* usado na indexação, porém aqui o nome do campo especifica onde será realizada a busca e o valor é a *String* que contém a palavra ou a frase a ser pesquisada.
- *Query* – é subclasse concreta que encapsula a lógica de um tipo de consulta particular, utilizada por *IndexSearcher* para realizar a pesquisa. Lucene dispõe de várias implementações de classes abstratas para realizar diferentes tipos de consultas.
- *TopDocs* – é a classe que armazena os *n* primeiros documentos resultantes de uma consulta, onde *n* é o numero parametrizado na classe *IndexSearcher.*

Lucene será mais bem detalhada no decorrer deste trabalho, visto que é a principal ferramenta utilizada para o desenvolvimento desta aplicação, por ser responsável por manter o índice com os documentos e realizar a busca aos mesmos.

3.2 DESCRIÇÃO DO FRAMEWORK

Como dito anteriormente, o foco deste trabalho é implementar um algoritmo de detecção de plágio utilizando a mesma estrutura do framework, baseado em Web Service RESTful, desenvolvido por JACHINSKI e GELINSKI (2011).

Baseado em conceito de Web Services RESTful, o trabalho de JACHINSKI e GELINSKI (2011) foi focado principalmente na implementação dos serviços e a utilização de conceitos como o RESTful e ferramentas como Lucene para a indexação e pesquisa de documentos. São dois os serviços REST desenvolvidos: o envio de arquivo para indexação e o envio de arquivo para a análise de detecção de plágio.

Os serviços REST desenvolvidos são implementados pelas classes ArquivoResource e PesquisaResource que são responsáveis pela indexação e pesquisa respectivamente. No serviço de indexação de arquivo é recebido como parâmetros o autor, título, tags, descrição e o caminho do arquivo de onde é realizado o upload do mesmo. Já no serviço de pesquisa recebe-se o diretório do arquivo de onde é feito upload.

A classe Lucene contém, basicamente, toda a implementação do algoritmo de busca, assim como indexação e demais funcionalidades providas pela biblioteca Lucene.

Figura 1 ilustra o diagrama de classe do framework desenvolvido.

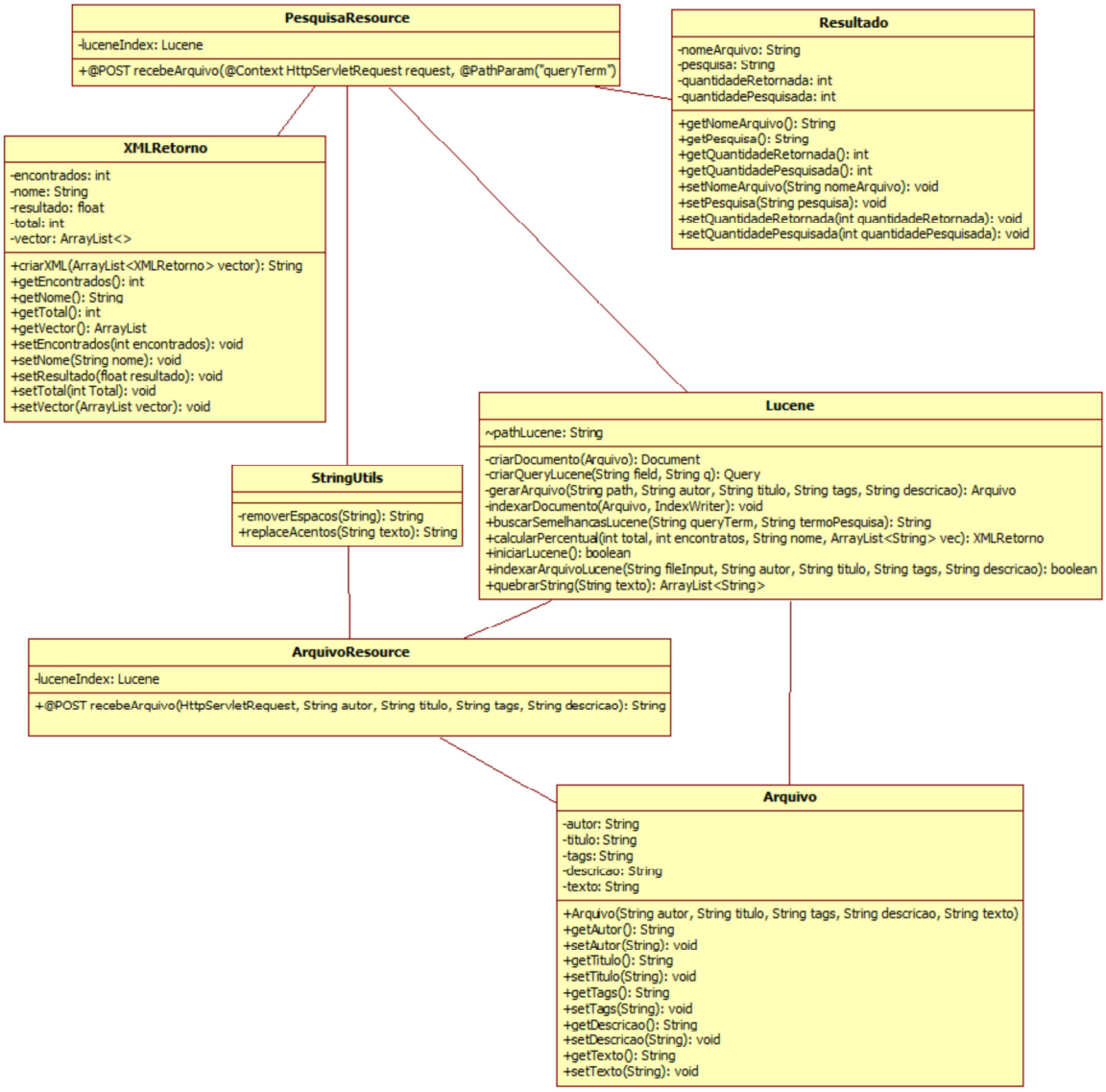


Figura 1 – Diagrama de Classe do Framework
Fonte: (JACHINSKI; GELINSKI, 2011).

Como explicado anteriormente na seção 2.3, tanto no serviço de indexação quanto no de pesquisa é gerado ao cliente, como retorno, um texto em formato XML. No serviço de indexação tem-se como retorno o *id* do arquivo e o nome do mesmo. Já no serviço de busca as informações recebidas como retorno são os documentos contidos no índice com o seus respectivos: *id*, *total*, *encontrados*, *percentual* e *nome*. Onde, *total* é a quantidade de frases usadas como pesquisa, *encontrados* é o total de frases que retornaram similaridades, o *percentual* é a quantidade de similaridades em relação ao *total* (JACHINSKI; GELINSKI, 2011).

Segundo JACHINSKI e GELINSKI (2011) o principal objetivo do framework é o relacionamento entre as ferramentas utilizadas e a implementação dos serviços

REST. Desta forma, o desenvolvimento de uma técnica de detecção de plágio mais eficiente foi sugerida como um trabalho futuro.

3.3 DESCRIÇÃO DA TÉCNICA UTILIZADA

Visando auxiliar no combate ao plágio, o foco do presente trabalho é implementar um algoritmo de detecção de plágio, mantendo a mesma estrutura de serviços REST do framework desenvolvido por JACHINSKI e GELINSKI (2011), de modo que auxilie ao docente na sua avaliação. O objetivo principal é pesquisar similaridades entre os textos melhorando os resultados mostrados ao cliente no final da análise.

A técnica aqui empregada visa encontrar, no índice, documentos que contenham fragmentos de texto idênticos ao documento analisado. Deste modo, cada trecho de texto encontrado no índice será identificado e retornado com o seu respectivo documento. Cabe ressaltar que o índice é uma base local de documentos indexados, portanto, textos que foram copiados de fontes, como a Internet, por exemplo, não serão o foco.

Como explicado anteriormente, um n-grama é uma sequência de *n* elementos de uma *String*. A técnica pode ser utilizada tanto para caracteres quanto para palavras. Por exemplo: a frase “ser ou não ser”, utilizando $n = 2$ na abordagem de palavras resulta nas seguintes sequências: “ser ou”, “ou não”, “não ser”.

Por ser difícil definir qual é a quantidade mínima de palavras sequenciais que caracterizam o plágio, optou-se que o valor de *n* será variável de acordo com o parâmetro, definido pelo usuário na aplicação cliente, recebido no serviço de pesquisa.

Cabe ressaltar que o valor de *n* para esta técnica, é apenas o número de palavras utilizadas para realizar a consulta. Neste caso, supondo que *n* é igual a cinco, é correto afirmar que as frases retornadas terão no mínimo cinco palavras.

3.4 ARQUITETURA DA APLICAÇÃO

Parte da arquitetura deste trabalho relacionada a implementação das funcionalidades da biblioteca Lucene como a indexação e pesquisa e o retorno XML ao cliente tiveram modificações relevantes para que se pudesse implementar o algoritmo proposto e obtivesse melhor desempenho. Quanto as classes relacionadas a implementação do serviço REST, modificaram-se apenas alguns parâmetros de entrada de forma a atender as necessidades da aplicação.

Como o principal objetivo do trabalho foi concentrado na técnica de detecção de plágio, detalhes relacionados à interface e usabilidade do sistema não foram o foco do desenvolvimento. Desta forma, a ferramenta aceita somente arquivos em formato PDF, por exemplo.

Com a reutilização dos serviços REST desenvolvidos no framework por JACHINSKI e GELINSKI (2011), o foco deste trabalho concentrou-se em aspectos de como melhorar a indexação, a busca e processamento de resultados e retorno dos mesmos.

A figuras 2 e 3 ilustram, respectivamente, as classe responsáveis por implementar os serviços REST de indexação e pesquisa e a figura 4 ilustra o diagrama de classes do restante da aplicação.

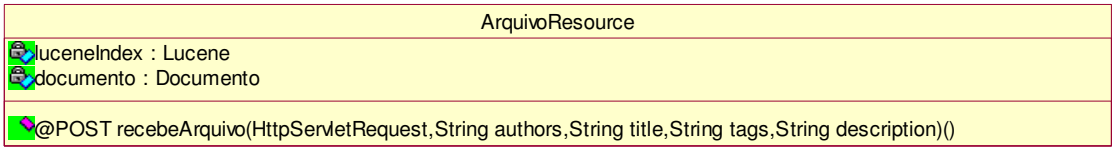


Figura 2 - Classe responsável pelo serviço de Indexação.
Fonte: Autoria própria.

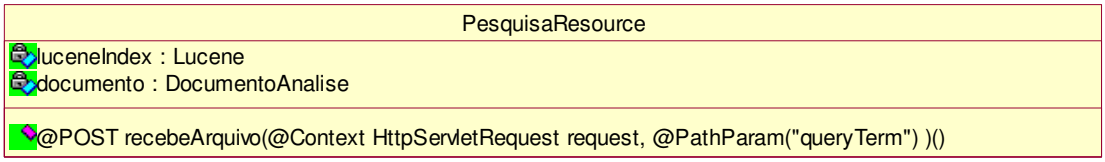


Figura 3 - Classe responsável pelo serviço de Pesquisa.
Fonte: Autoria própria.

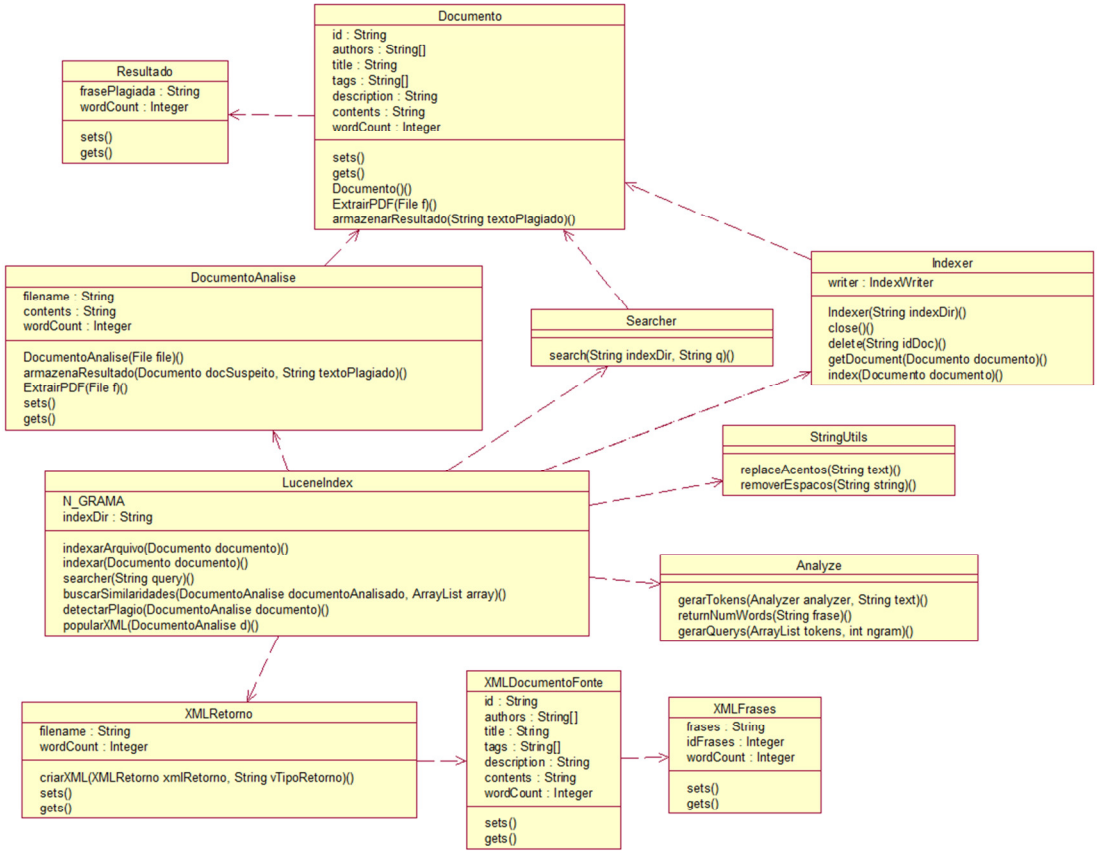


Figura 4 – Diagrama de Classe da Aplicação
Fonte: Autoria própria

Umas das importantes etapas do trabalho foi definir como seria representado cada documento, visto que o mesmo deveria ser suficientemente flexível de modo que pudesse ser utilizado na indexação quanto no retorno da busca e também no processamento dos resultados. Desta forma criou-se a classe **Documento** [1] definindo os seguintes atributos:

- *id*: identificação única para cada documento indexado. O valor do id é gerado pelo método *randomUUID()* da classe *java.util.UUID* que consiste num identificador universal exclusivo;
- *authors*: vetor de *String* que contém os autores do documento;
- *title*: título do documento;

[1] As classes que estão destacadas em negrito referem-se às classes implementadas na ferramenta. As classes em itálico referem-se àquelas da ferramenta Lucene.

- *tags*: vetor para a utilização livre de termos que de uma forma possam categorizar ou classificar o documento indexado para possibilitar a filtragem na etapa da busca;
- *description*: descrição do documento;
- *contents*: contém todo o texto do documento que será indexado para posterior análise;
- *wordCount*: armazena o número total de palavras contidas no documento.

A classe **Documento** também é responsável por extrair o texto do arquivo em PDF. Para realizar a extração do texto é utilizada a biblioteca *PDFTextStripper*, já utilizada no trabalho anterior.

No Lucene para que cada arquivo seja indexado, antes ele é representado pela classe *Document* que contém um ou mais *Field*, ou seja, o documento com os seus respectivos atributos. Cada *Field* é composto por uma *String* que o identifica, o valor do mesmo, valor booleano que define se o campo será armazenado e o valor booleano que especifica se o campo será analisado. A forma de como o texto é indexado e posteriormente pesquisado depende muito de como cada *Field* foi especificado.

Por exemplo, o quadro 1 mostra a criação de um documento com dois atributos. Onde o conteúdo do *field1* será armazenado e não analisado e o *field2* terá o seu conteúdo não armazenado, mas será analisado. A análise do texto na Lucene é uma importante e necessária etapa, que será mais bem detalhada logo adiante.

```
Document d = new Document();

d.add(new Field("field1", field1, Field.Store.YES, Field.Index.NOT_ANALYZED);

d.add(new Field("field2", field2, Field.Store.NO, Field.Index.ANALYZED);
```

Quadro 1 – Exemplo de criação de um objeto Document e seus respectivos Fields.

Aqui neste trabalho, cada documento indexado no índice do Lucene será representado da mesma forma que foi especificado na classe **Documento**, porém, com exceção de que no Lucene será criado um atributo (*Field*) extra para armazenar o conteúdo do documento com a remoção das *stop words*.

Como dito anteriormente, uma das importantes etapas antes da indexação é a análise e processamento do texto de cada um dos *Field* do documento. Onde cada palavra é extraída em um vetor de termos para então ser indexada. A classe que realiza tal função na Lucene é a *Analyzer*.

Algumas implementações da classe *Analyzer*, disponibilizadas pelo Lucene, também oferecem funcionalidades como remoção de palavras comuns que não possuem grande relevância (*stop words*), a redução das palavras às suas raízes (*stemmer*) e a utilização de sinônimos como opção de pesquisa. Inicialmente implementadas somente para a língua inglesa, atualmente tem-se disponíveis classes *Analyzer* para diversos idiomas, como a *BrazilianAnalyzer* para o português brasileiro.

Lucene oferece a opção de utilizar mais de um *Analyzer* para diferentes *Field* de um *Document*. Para tal, é necessário utilizar a classe *PerFieldAnalyzerWrapper*, onde é especificado o *Analyzer default* que será utilizado, desta forma o Field que utilizará uma diferente implementação de *Analyzer* terá de ser especificado. O quadro 2 mostra como as duas implementações utilizadas neste trabalhos são especificadas, desta forma todos os *Field* do documento a ser indexado terão como o *Analyzer* o *WhitespaceAnalyzer*, exceto “content2” que utilizará o *StandardAnalyzer*.

```
PerFieldAnalyzerWrapper analyzer = new PerFieldAnalyzerWrapper(new WhitespaceAnalyzer(Version.LUCENE_32));

analyzer.addAnalyzer("content2", new StandardAnalyzer(Version.LUCENE_32, analyze.getStopwords()));
```

Quadro 2 – Diferentes implementações da classe Analyzer para cada Field.

Como a técnica de detecção de plágio aqui utilizada está focada em encontrar trechos de texto idênticos em diferentes documentos, escolheu-se a classe *WhitespaceAnalyzer,* a qual tem como limitador para cada termo (palavra) o espaço em branco. Contudo, também será disponibilizada a opção da análise com o texto sem as *stop words* (palavras irrelevantes) através da classe *StandardAnalyzer*. Destaca-se que a opção de trabalhar com *StandardAnalyzer* ocorreu pelo fato de que a *BrazilianAnalyzer* não está suficientemente madura para atender as necessidades deste trabalho.

Depois de analisado o texto de cada *field* do documento, a próxima etapa consiste na indexação. A forma em que o processo de indexação é realizado nesta aplicação é mostrada pelo diagrama de sequência ilustrado na figura 5.

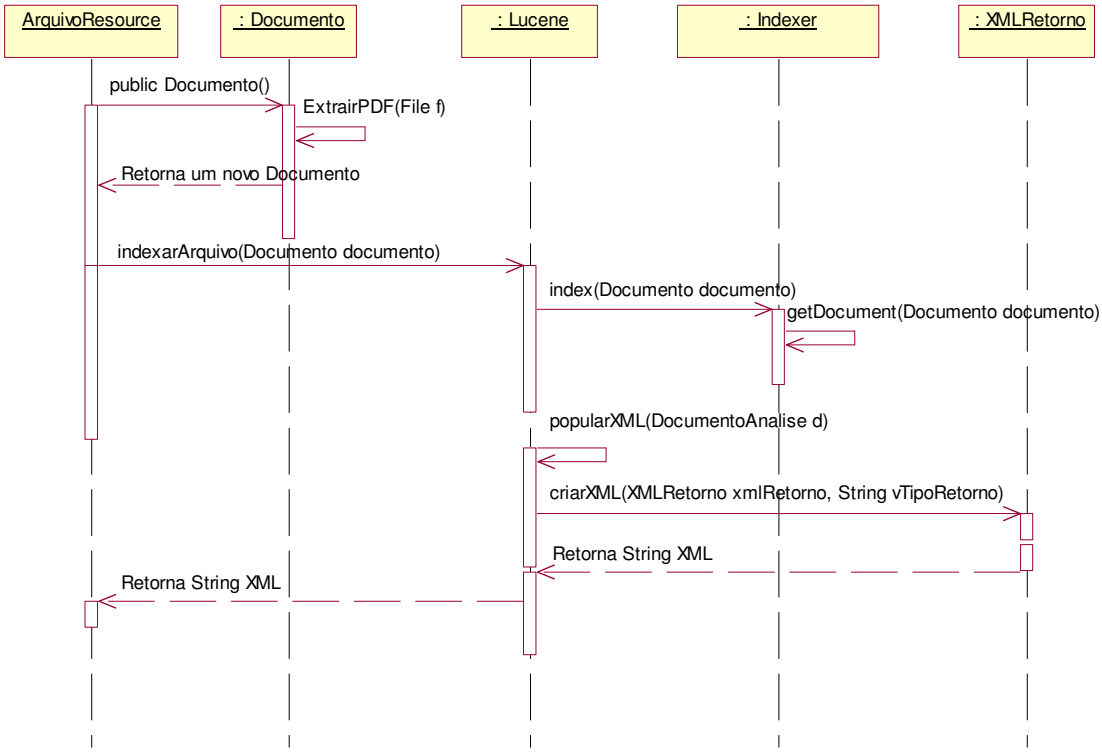


Figura 5 – Diagrama de sequência da Indexação
Fonte: Autoria própria

Conforme ilustra o diagrama (figura 5), na etapa da indexação o documento que é enviado pela aplicação cliente e recebido pela classe **ArquivoResource**, passa a ser representado pelo objeto documento da classe **Documento**, de onde é realizada a extração do texto contido no arquivo PDF. No próximo passo o documento é enviado ao método indexarArquivo() da classe **LuceneIndex**.

LuceneIndex é a classe central que intermedia as funcionalidades da biblioteca Apache Lucene implementadas pela classe **Indexer**, **Analyze** e **Searcher**. Na classe **LuceneIndex** instancia-se um objeto da classe **Indexer** especificando o diretório do índice e passando o documento a ser indexado.

Após criado o documento com os seus atributos e realizada a configuração necessária, a classe **Indexer** realiza a indexação propriamente dita e como resposta ao serviço é gerado um retorno em formato XML, implementado na classe **XMLRetorno**. A figura 6, abaixo, ilustra um exemplo de retorno de um arquivo indexado.

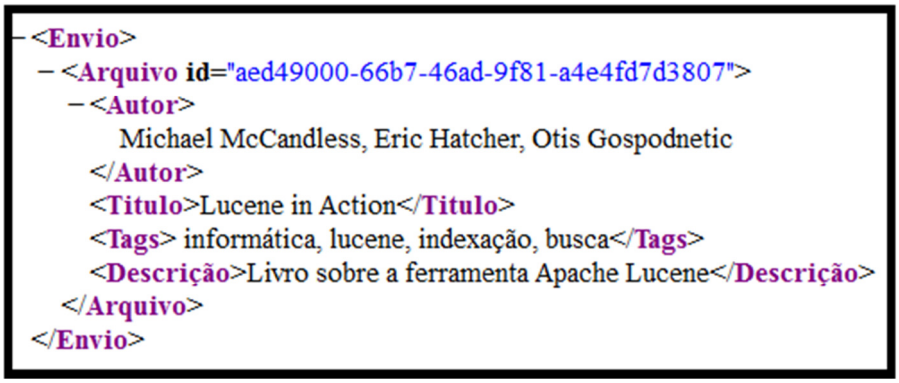


Figura 6 – Retorno XML do serviço de indexação.
Fonte: Autoria própria

Na etapa da pesquisa, a principal classe que implementa a busca ao índice do Lucene é a *IndexSearcher*. Através da classe *Directory*, responsável por abrir o diretório onde se encontra o índice, pode-se dizer que *IndexSearcher* é a que permite o acesso a leitura ao mesmo (MCCANDLESS; HATCHER; GOSPODNETIC, 2010).

Antes de realizar a busca ao índice deve-se entender primeiramente qual será o tipo de consulta a ser utilizada. Lucene oferece diferentes opções de consulta que são implementadas pela classe *Query* e suas subclasses, dentre as principais estão:

- *TermQuery*: é o método mais comum de consulta, onde realiza-se a pesquisa através de um termo exato;

  query = new TermQuery(new Term(“contents”, “plagio”));

  Onde o resultado serão os documentos que contém a palavra plagio no *field* “contents”.

- *PhraseQuery*: utilizada para pesquisar por frases específicas no índice do Lucene. Através do método *setSlop()* é possível configurar a distância desejada entre uma palavra e outra.

  PhraseQuery query = new PhraseQuery();
  query.setSlop(0);
  query.add(new Term("contents", “apache”));
  query.add(new Term("contents", “lucene”));

Neste caso serão retornados os documentos que possuírem exatamente a frase “apache lucene” no *field* “contents”.

- *FuzzyQuery*: similar ao algoritmo de edição de distância de *Leveshtein Distance*, descrito anteriormente, tem como objetivo pesquisar por termos similares.

query = new FuzzyQuery(new Term(“contents”, “plágeo”));

Mesmo estando errada a palavra a ser consultada, quando for executada esta *query* em um índice que contenha a palavra plágio no *field* “contents” retornará resultados.

Neste trabalho foi escolhido trabalhar com *PhraseQuery*, pois é a única que utiliza informações sobre a posição de cada termo que está contido no índice do Lucene. Desta forma, como o objetivo é encontrar frases idênticas, este tipo de consulta é a mais adequada para este propósito.

O valor utilizado para determinar a distância entre as palavras, através do método *setSlop(),* será zero para o texto normal, onde o propósito é encontrar documentos que contenham frases idênticas às consultadas. E para o texto que utiliza o processamento de *stop words*, o valor será cinco, pois quando a Lucene remove as palavras irrelevantes, as posições das palavras que permanecem não são atualizadas, desta forma como é difícil prever a quantidade de exclusões que serão feitas entre uma palavra e outra, optou-se por definir este número por ser razoável.

Além do tipo de consulta utilizado, Lucene oferece métodos de busca que possibilitam a filtragem nos resultados. A classe abstrata utilizada para isto é *Filter*, que possui várias implementações disponíveis pela biblioteca. Neste trabalho foi utilizada a classe *TermsFilter*, que possibilita a filtragem da consulta por documentos que contenham determinado(s) termo(s). Foi escolhida esta classe, por ser a única a suportar vários termos como filtro.

As opções de filtro disponibilizadas aqui são por autor e por tags, sendo que é possível definir várias palavras como filtragem, porém deve-se escolher apenas uma das opções para cada consulta.

Tendo definido o tipo de consulta a ser utilizada na pesquisa, é necessário então, utilizar a classe *TopDocs* para armazenar os resultados encontrados pelo

método de busca do *IndexSearcher.* O acesso a cada um dos resultados, ou seja, aos documentos encontrados na pesquisa, é provido pela classe *ScoreDocs*. Através da classe *ScoreDocs* é possível acessar os mesmos *fields* que foram definidos quando feita a indexação para cada *Document* que a busca retornar.

3.5 IMPLEMENTAÇÃO DO ALGORITMO

Como descrito anteriormente a técnica de detecção de plágio aqui implementada é baseada na abordagem n-grama e visa identificar trechos textuais idênticos em documentos distintos.

Basicamente, o algoritmo aqui proposto concentra-se principalmente na forma de organização dos resultados retornados pela busca realizada ao índice da biblioteca Lucene. Levando em conta a possibilidade de que se tenha um grande número de documentos indexados e que para cada consulta realizada poderão ser retornados vários documentos como resultado, destaca-se a necessidade de organização e apresentação de resultados de maneira adequada e precisa.

As seguintes etapas são necessárias para descobrir se há suspeitas de plágio em um determinado documento:

- A extração do texto em um vetor de palavras;
- A geração das frases que serão pesquisadas;
- A pesquisa de cada frase;
- O armazenamento dos resultados de cada frase.

Conforme o diagrama de atividades do algoritmo, ilustrado na figura 7, a tarefa de detecção de plágio inicia-se com a extração e armazenamento das palavras do texto em um vetor. A extração é implementada pelo método gerarTokens() da classe **Analyze**.

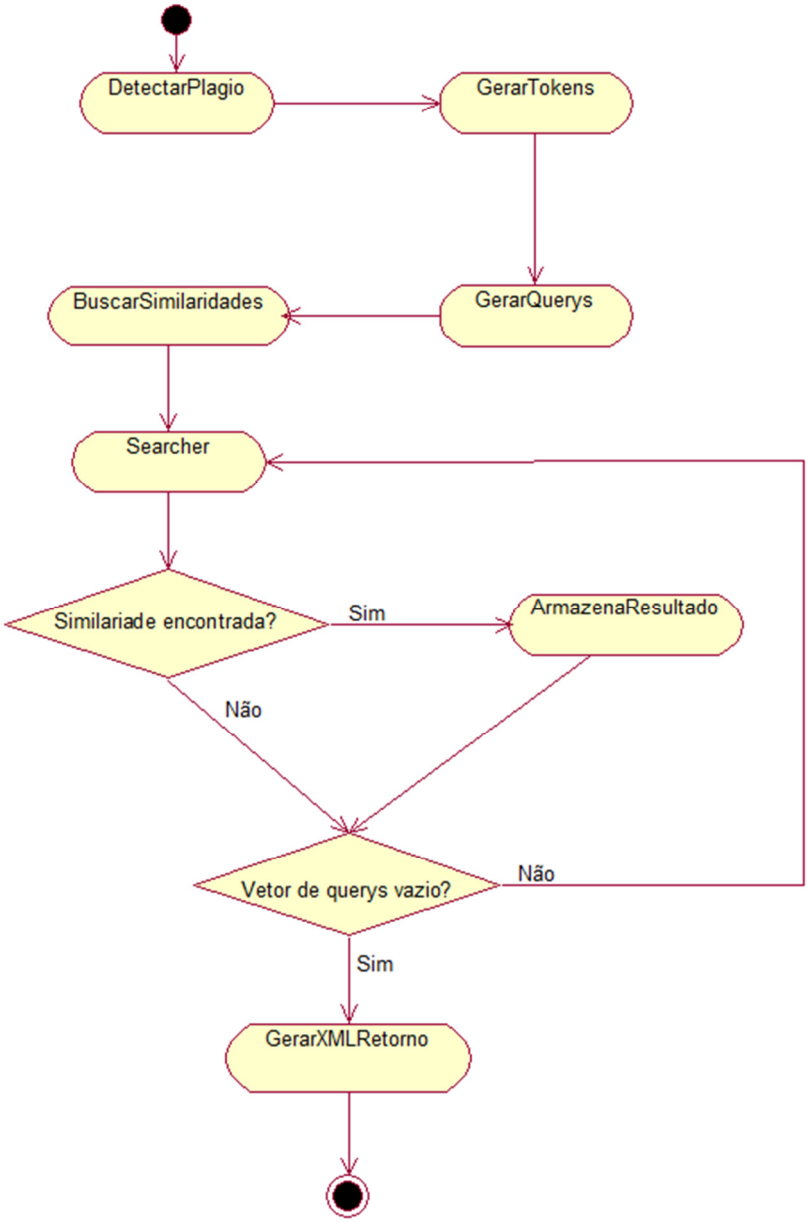


Figura 7 – Diagrama de atividades do algoritmo
Fonte: Autoria própria

A classe *Analyzer* provida pelo Lucene e utilizada na indexação, também permite o acesso a cada termo (palavra) do texto que está sendo analisado. Desta forma, escolheu-se utilizar a mesma classe para extração das palavras do texto em um vetor. A composição de cada vetor dependerá do tipo do *Analyzer* utilizado. Como explicado anteriormente, serão utilizados dois tipos de *Analyzer*: o *WhitespaceAnalyzer* que tem como delimitador de termos os espaço em branco e *StandardAnalyzer* para o processamento do texto sem as *stop words* (palavras consideradas irrelevantes). A opção de utilizar *stop words* é especificada como um dos parâmetros de chamada do serviço de pesquisa.

Na próxima etapa são geradas, em um vetor, as frases que serão utilizadas na consulta. O número de palavras contidas em cada frase varia de acordo com o parâmetro N_GRAMA definido na aplicação cliente.

Para cada frase é realizada a pesquisa no índice, onde o resultado será um vetor dos documentos que contém a frase, caso não exista resultado o retorno é nulo. A figura 8 mostra a relação entre as classes envolvidas no processamento do resultado.

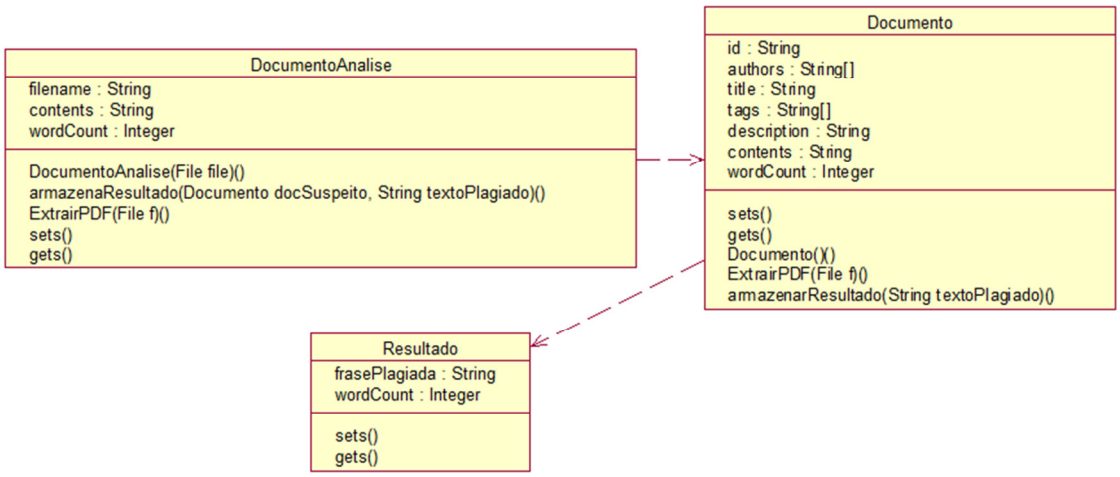


Figura 8 – Classes utilizadas no processamento de resultados
Fonte: Autoria própria

Conforme mostra a figura 8, o documento analisado possui um vetor de documentos suspeitos de plágio, que por sua vez podem conter vários trechos de texto, representados pela classe **Resultado**.

Desta forma, quando realizada a busca e retornado o resultado, para cada documento suspeito é verificado se a frase encontrada é a continuação de um resultado anterior ou é o início de um novo resultado.

Ao final do processo é gerado, em formato XML, o resultado da análise contendo os documentos suspeitos e as sequências de textos encontradas em cada um deles.

O relatório de retorno é composto pelo documento analisado mostrando o total de palavras contidas no mesmo e os documentos fontes com seus respectivos *id*, *authors*, *title*, *tags*, total de palavras e as frases encontradas. Para cada frase também mostra-se a quantidade de palavras contidas na mesma. A figura 9 ilustra um exemplo de relatório da uma análise retornado ao usuário.

Cabe ressaltar que a ideia de apresentar a quantidade de palavras tanto no documento analisado quanto no documento fonte e nas frases é possibilitar que aplicações clientes possam realizar diferentes interpretações dos dados apresentados, como por exemplo, pode-se calcular o percentual de palavras semelhantes em relação ao total do texto.


```
<Resultados>
 <DocumentoAnalisado File="Doc.pdf" Total_Palavras_Documento_Analisado="58">
  <DocumentoFonte id="aed49000-66b7-46ad-9f81-a4e4fd7d3807" authors=" Michael
   McCandless, Eric Hatcher, Otis Gospodnetic" title="Lucene in Action" tags=" informática,
   lucene, indexação, busca" Total_palavras_Documento="178985">
   <frases id="0" Total_Palavras_Frase="26">
     field cache may consume quite a bit of memory; each entry allocates an array of the native
     type, whose length is equal to the number of
   </frases>
   <frases id="1" Total_Palavras_Frase="29">
     in the provided reader. The field cache doesn't clear its entries until you close your reader
     and remove all references to that reader from your application and garbage collection
   </frases>
  </DocumentoFonte>
 </DocumentoAnalisado>
</Resultados>
```


Figura 9 – Relatório da análise retornado ao usuário.
Fonte: Autoria própria

3.6 TESTES

Os testes aqui realizados têm como objetivo avaliar a implementação do algoritmo de detecção de plágio, mostrando a consistência dos resultados. Fatores como o tempo de processamento, o consumo de memória e outros testes de desempenho e segurança não são o principal objetivo desta pesquisa. Porém, em MCCANDLESS (2010) *et al*, os autores destacam a eficiência e rapidez da Lucene na realização das buscas e o suporte para grande volume de dados indexados. Como será a biblioteca, responsável pela parte mais crítica desta aplicação não houve testes mais aprofundados nestes quesitos.

Cabe ressaltar também que neste trabalho o objetivo não é a interpretação dos resultados, mas sim apenas a apresentação. Desta forma, cabe ao docente avaliar os resultados de cada documento.

Para os testes aqui realizados, sem a opção de *stop words*, optou-se por definir o valor de *n* igual a cinco para n-grama, pois, como descrito anteriormente, o valor de *n* é apenas o número de palavras sequenciais contidas em uma frase para realizar a consulta.

Para a realização dos testes foram escolhidas três monografias utilizadas como referência neste trabalho. Desta forma, o texto a ser analisado conterá parágrafos idênticos de cada um dos seguintes documentos indexados:

- Documento 1:
  authors: Anderson B. Jachinski, Rafael Gilinski;
  title: Ferramenta Baseada em Web Service RESTful para Indexação e Verificação de Plágio em Trabalhos Acadêmicos;
  tags: informática, monografia, tcc;
  description: Trabalho de Conclusão de Curso.

- Documento 2:
  authors: Rodrigo M. de Abreu;
  title: Proposta de arquitetura para um Sistema de Detecção de Plágio Multi-algoritmo;
  tags: informática, dissertação, mestrado;
  description: Dissertação de Mestrado.

- Documento 3:
  authors: Flávia A. de O. Santos;
  title: Criação de Ferramenta de Detecção de Plágio em Ambiente Virtual de Aprendizagem;
  tags: informática, mestrado, dissertação;
  description: Dissertação de Mestrado.

O documento que será usado para a análise foi construído de forma a conter parágrafos de cada um dos documentos indexados, intercalados com parágrafos de outro texto qualquer.

Na primeira consulta efetuada sem realizar qualquer tipo de filtro e sem utilizar as *stop words* o resultado mostrou-se satisfatório, pois todos os parágrafos copiados foram retornados com os seus respectivos documentos. Na consulta

utilizando a filtragem pelos termos “tcc” e “monografia” no campo *tags* o resultado, ilustrado na figura 10, também se mostrou positivo, visto que foram retornados somente os documentos que realmente continham estes termos.


```
<Resultados>
  <DocumentoAnalisado File="teste.pdf" Total_Palavras_Documento_Analisado="737">
    <DocumentoFonte id="f0c11519-123a-4077-8735-ed30e8846779" authors=" Anderson B. Jachinski, Rafael Gelinski" title="Ferramenta Baseada em Web Service RESTful para Indexação e Verificação de Plágio em Trabalhos Acadêmicos" tags=" tcc, monografia, informática, plágio" Total_palavras_Documento="15586">
      <frases id="0" Total_Palavras_Frase="45">
        a evolução tecnológica, criaramse inúmeras facilidades para a humanidade. Dentre elas, a Computação e a Internet se destacam por reduzir custos, distância e tempo para trafegar inúmeros tipos de informação. Porém, isso também permitiu o crescimento de alguns pontos negativos em relação à propriedade intelectual.
      </frases>
    </DocumentoFonte>
    <DocumentoFonte id="675d05ee-6372-49a5-9a4c-0df809cfd026" authors=" Flávia A. O. Santos" title="Criação de Ferramentas de Detecção de Plágio em Ambiente Virtual de Aprendizagem" tags=" informática, plágio, monografia, tcc" Total_palavras_Documento="14506">
      <frases id="0" Total_Palavras_Frase="77">
        Perrenoud 1999, a avaliação da aprendizagem, é um processo mediador na construção do currículo e se encontra intimamente relacionada à gestão da aprendizagem dos alunos. Na avaliação da aprendizagem, o professor não deve permitir que os resultados das provas periódicas, geralmente de caráter classificatório, sejam supervalorizados em detrimento de suas observações diárias, de caráter diagnóstico. O professor, que trabalha numa dinâmica interativa, tem noção, ao longo de todo o ano, da participação e produtividade de cada aluno.
      </frases>
      <frases id="1" Total_Palavras_Frase="78">
        A avaliação formativa exige um monitoramento contínuo da participação dos alunos e geração de feedback aos mesmos. Percebese então, que a quantidade de variáveis a serem consideradas para a eficácia de uma avaliação formativa gera uma sobrecarga de trabalho para o professor podendo comprometer e até inviabilizar o processo de avaliação formativa em um curso. Constituise, portanto, um desafio desenvolver tecnologia que dê suporte a este processo de avaliação, facilitando a tarefa do professor no desenvolvimento das atividades.
      </frases>
    </DocumentoFonte>
  </DocumentoAnalisado>
</Resultados>
```

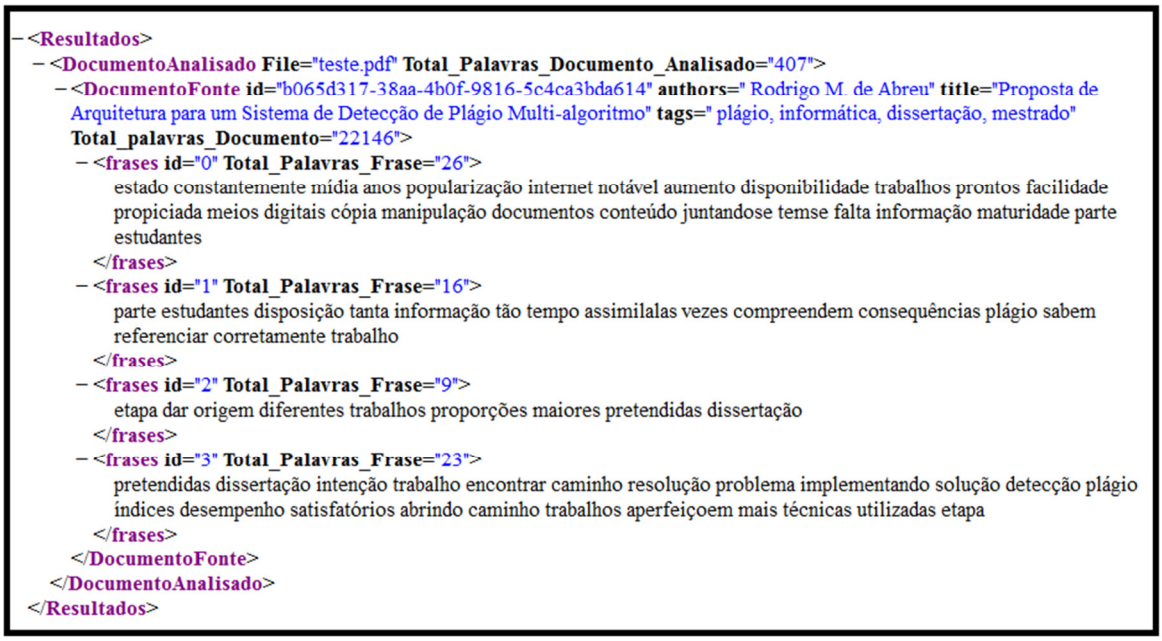

Figura 10 – Relatório de consulta utilizando filtro tags: “tcc” e “monografia”
Fonte: Autoria própria

Na pesquisa realizada utilizando o filtro pela palavra “mestrado” no campo *tags* sem utilizar as *stop words*, o resultado obtido é mostrado na figura 11.


```
<Resultados>
  <DocumentoAnalisado File="teste.pdf" Total_Palavras_Documento_Analisado="737">
    <DocumentoFonte id="b065d317-38aa-4b0f-9816-5c4ca3bda614" authors=" Rodrigo M. de Abreu" title="Proposta de Arquitetura para um Sistema de Detecção de Plágio Multi-algoritmo" tags=" plágio, informática, dissertação, mestrado" Total_palavras_Documento="22146">
      <frases id="0" Total_Palavras_Frase="85">
        plágio tem estado constantemente na mídia nos últimos anos. Com a popularização da Internet, é notável o aumento da disponibilidade de trabalhos prontos e a facilidade propiciada pelos meios digitais para cópia e manipulação de documentos e de seu conteúdo. Juntandose a isso, temse a falta de informação e maturidade por parte dos estudantes, que tem a sua disposição tanta informação e tão pouco tempo para assimilalas, que por muitas vezes não compreendem as consequências do plágio, ou mesmo sabem como referenciar corretamente um trabalho.
      </frases>
      <frases id="1" Total_Palavras_Frase="62">
        Cada etapa por si só pode dar origem a diferentes trabalhos, de proporções até maiores do que as pretendidas nesta dissertação, porém a intenção deste trabalho é de encontrar um caminho para resolução do problema, implementando uma solução para detecção de plágio com índices de desempenho satisfatórios e abrindo caminho para trabalhos que aperfeiçoem ainda mais as técnicas utilizadas em cada etapa.
      </frases>
    </DocumentoFonte>
  </DocumentoAnalisado>
</Resultados>
```


Figura 11 – Relatório de consulta utilizando filtro tags: “mestrado”
Fonte: Autoria própria

Nos testes realizados com a utilização da técnica *stop words,* observou-se que os resultados variavam de acordo com o valor de *n* para n-grama. Isto se deve ao fato de que as posições das palavras do texto indexado não seguem uma sequência, como explicado anteriormente. Deste modo, quanto menor o valor de *n*, mais aproximado será o resultado. A figura 12 mostra o resultado obtido através da consulta realizada com a filtragem *tags* = “monografia”, utilizando a técnica *stop words* e atribuindo *n* = 3 para n-grama.



```
- <Resultados>
  - <DocumentoAnalisado File="teste.pdf" Total_Palavras_Documento_Analisado="407">
    - <DocumentoFonte id="b065d317-38aa-4b0f-9816-5c4ca3bda614" authors=" Rodrigo M. de Abreu" title="Proposta de
      Arquitetura para um Sistema de Detecção de Plágio Multi-algoritmo" tags=" plágio, informática, dissertação, mestrado"
      Total_palavras_Documento="22146">
      - <frases id="0" Total_Palavras_Frase="26">
          estado constantemente mídia anos popularização internet notável aumento disponibilidade trabalhos prontos facilidade
          propiciada meios digitais cópia manipulação documentos conteúdo juntandose temse falta informação maturidade parte
          estudantes
        </frases>
      - <frases id="1" Total_Palavras_Frase="16">
          parte estudantes disposição tanta informação tão tempo assimilalas vezes compreendem consequências plágio sabem
          referenciar corretamente trabalho
        </frases>
      - <frases id="2" Total_Palavras_Frase="9">
          etapa dar origem diferentes trabalhos proporções maiores pretendidas dissertação
        </frases>
      - <frases id="3" Total_Palavras_Frase="23">
          pretendidas dissertação intenção trabalho encontrar caminho resolução problema implementando solução detecção plágio
          índices desempenho satisfatórios abrindo caminho trabalhos aperfeiçoem mais técnicas utilizadas etapa
        </frases>
      </DocumentoFonte>
    </DocumentoAnalisado>
  </Resultados>
```


Figura 12 – Resultado obtido utilizando *stop words*.
Fonte: Autoria própria.

Pode-se observar, através das figuras 11 e 12, a diferença dos resultados obtidos entre os dois tipos de consultas utilizadas, onde a figura 12 mostra que os resultados da análise utilizando *stop words* retornam mais frases, porém com menos palavras.

Como explicado anteriormente, a técnica aqui implementada está focada na detecção de textos idênticos, desta forma a técnica *stop words* foi apenas uma opção incremental para que trabalhos futuros possam melhorá-la e adequá-la o seu uso.

3.7 PROBLEMAS ENCONTRADOS

Como o propósito deste trabalho foi implementar funcionalidades baseando-se em um framework já desenvolvido, foi preciso primeiramente entender a arquitetura e lógica da aplicação e quais os conceitos e ferramentas que foram utilizadas. Uma das dificuldades encontradas no início foi o entendimento de como funciona uma aplicação que segue conceitos REST. Pois, apesar de neste trabalho ser mantido os mesmos serviços implementados no framework desenvolvido, foi necessário realizar algumas mudanças em relação a alguns parâmetros, tanto na chamada ao serviço de indexação quanto ao de busca. Além disso, foi preciso implementar um serviço extra para possibilitar o download do documento suspeito através de seu *id*.

Em relação à biblioteca Lucene que, apesar de sua simplicidade no uso, foi necessário aprofundar no estudo de como os documentos seriam indexados e analisados para que posteriormente na pesquisa fossem recuperados de maneira esperada. Disponibilizar a opção de filtragem na consulta, também foi uma tarefa uma tanto custosa, visto que foi necessário entender quais filtros disponibilizados pela biblioteca são adequados a determinada situação.

Na implementação do algoritmo, a preocupação foi de como organizar os resultados obtidos para cada consulta. Pois, considerando que o índice utilizado pode conter um grande volume de documentos indexados, foi necessário pensar em uma maneira de organizá-los de forma eficiente e consistente, de forma que fossem corretos e confiáveis.

4 CONCLUSÃO

A utilização da biblioteca Lucene para indexação e recuperações de informações aliada a abordagem n-grama, aqui desenvolvida, mostraram-se eficientes no propósito de encontrar similaridades entre diferentes textos. A forma de como os resultados da busca são organizados na aplicação desenvolvida, permite que a consulta em uma base indexada, que mesmo contendo uma grande quantidade de documentos, possa retornar resultados consistentes e satisfatórios.

As diferentes ferramentas e conceitos aqui utilizados foram de grande importância para adquirir conhecimento através deste trabalho. Como a implementação realizada neste trabalho foi baseada no framework desenvolvido por JACHINSKI e GELINSKI (2011), foi necessário antes, entender as ferramentas para então realizar os ajustes de forma que atendesse a necessidade da aplicação. Desta forma, também pode-se adquirir conhecimentos através destas ferramentas.

Apesar da técnica aqui empregada, ser umas das mais simples formas de detecção de plágio. Foi um passo importante e necessário para auxiliar acadêmicos que queiram contribuir com funcionalidades que objetivem a melhorar e estender mecanismos que auxiliem a detectar diferentes tipos de plágio.

4.1 TRABALHOS FUTUROS

Devido à forma em que se encontra a estrutura do framework, existe à possibilidade de desenvolvimento de trabalhos futuros, os quais podem incrementar funcionalidades ou melhorar aquelas já existentes.

As diferentes formas de plágio praticadas por plagiários podem ser detectadas através da implementação de cada um dos métodos de detecção, citados na seção 2.1. Desta forma cada método poderia gerar um novo trabalho, ou então seria possível também desenvolver um trabalho que utilizasse várias destas técnicas. A biblioteca Lucene possui vários recursos que podem ser úteis, por exemplo, a opção de reduzir as palavras na sua raiz, a utilização das *stop words* e a possibilidade de incluir os sinônimos são funcionalidades que ajudariam a detectar similaridades em textos que possuem sintaxes diferentes. Também a ferramenta

WordNet, que disponibiliza uma base de recursos lexicais, pode ser utilizada em conjunto como a Lucene de forma a detectar similaridades semânticas.

Estender a consulta a motores de busca, como por exemplo, o Google, possibilitaria a identificação de plágio que tenham como fonte textos e documentos disponíveis na Internet.

Entre as melhorias que podem ser desenvolvidas estão possibilitar que a aplicação suporte também diferentes formatos de arquivos de entrada, além de PDFs, e implementar mecanismos mais eficientes que evitem a indexação de um documento já indexado.

A forma em que estão estruturados os serviços REST, é possível desenvolvimento de aplicações clientes em diferentes linguagens e plataformas de modo a estender ao alcance da ferramenta. Uma das possibilidades seria a integração com ambientes de aprendizagem virtual, como por exemplo, o MOODLE, onde poderia ser desenvolvido um módulo responsável por fazer a requisição dos serviços e apresentação dos resultados, assim como melhorar a interpretação dos mesmos através de gráficos que mostrem o percentual de palavras encontradas em cada documento.

REFERÊNCIAS

ABREU, Rodrigo M. **Proposta de Arquitetura para um Sistema de Detecção de Plágio Multi-algoritmo**. 2011. 105 f. Dissertação (Mestrado) – Programa de Pós-graduação em Engenharia de Sistemas e Computação, Universidade Federal do Rio de Janeiro. Rio de Janeiro 2011.

ALMEIDA, Marcos P. de; FERREIRA, Aline M. **Correção Automática de Palavras em textos**. 2004. Disponível em: <http://homepages.dcc.ufmg.br/~nivio/cursos/pa04/seminarios/seminario11/seminario11.html>. Acesso em: 14 de março de 2012.

ALZAHRANI, Salha; SALIM Naomie; AJITH, Abraham. **Understanding Plagiarism Linguistic Patterns, Textual Features and Detection Methods.** 2011.

AMJAD Al-Tobi; MOHAMED Elhadi. **Duplicate Detection in Documents and WebPages using Improved Longest Common Subsequence and Documents Syntactical Structures.** 2009.

COSTELLO, Roger L. **Building Web Services the REST Way**. Disponível em: <http://www.xfront.com/REST-Web-Services.html> Acesso em: 11 de Abril de 2012.

FIELDING, Roy Thomas. **Architetural Styles and the Design of Network-based Software Architectures**. 2000. 180 f. Dissertação (Doutorado) – Information and Computer Science, University of California. Irvine. 2000.

FIELDING, Roy T.; TAYLOR, Richard N. **Principled Design of the Modern Web Architetue.** 2010. Information and Computer Science, University of California. Irvine. 2000.

GALVÃO, Aline de O.; OLIVEIRA, Lucas D. **Avaliação de ferramentas para identificação de plágio em trabalhos acadêmicos.** 2011.

JACHINSKI, Anderson B.; GELINSKI, Rafael. **Ferramenta Baseada em Web Service RESTful para Indexação e Verificação de Plágio em Trabalhos Acadêmicos**. 2011. 63 f. Trabalho de Conclusão de Curso – Tecnologia em Análise de Desenvolvimento de Sistemas, Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2011.

JOSUTTIS, Nicolai M. **SOA In Practise**. Beijing, Cambridge, Farnham, Köln, Paris, Sebastopol, Taipei, Tokyo: O'Reilly 2007

MCCANDLESS, Michael; HATCHER, Erik; GOSPODNETIĆ, Otis**. Lucene in Action.** 2nd. Edition. Stamford: Manning, 2010.

OLIVEIRA, Tais da V. 2010. 31 f. **Adaptação do Algortimo Levenshtein Distance para o Cálculo de similaridade entre Frases**. 2010.

PIKE, R. **The Sherlock Plagiarism Detector**. Disponível em: <http://www.cs.su.oz.au/~scilect/sherlock>. Acesso em: 6 de junho de 2012.

ROMANCINI, Richard. 2007. **A praga do Plágio Acadêmico**. Disponível em: < http://sites.google.com/site/richardromancini/pragaplagio>. Acesso em 18 de maio de 2012.

SANDOVAL, Jose. **RESTful Java Web services**. Birmingham: Packt Publishing, 2009.

SANTOS, Flavia A. Oliveira. 2010. 75 f. **Criação da Ferramenta de Detecção de Plágio em ambiente Virtual de Aprendizagem**. Dissertação (Mestrado) – Programa de Pós-graduação em Engenharia Elétrica, Universidade Federal de Itajubá. Itajubá 2010.

WORDNET, 2012. **Sobre a WordNet.PT**. Disponível em: <http://www.clul.ul.pt/clg/wordnetpt/index.html>. Acesso em: 28 de março de 2012.