UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

KAIO PABLO GOMES

APLICAÇÃO DE BIOINFORMÁTICA PARA RECONHECIMENTO DE PLÁGIO EM CÓDIGOS DE PROGRAMAÇÃO

TRABALHO DE CONCLUSÃO DE CURSO

PONTA GROSSA
2017

KAIO PABLO GOMES

APLICAÇÃO DE BIOINFORMÁTICA PARA RECONHECIMENTO DE PLÁGIO EM CÓDIGOS DE PROGRAMAÇÃO

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Bacharel em Ciência da Computação, do Departamento Acadêmico de Informática, da Universidade Tecnológica Federal do Paraná.

Orientadora: Profª. Drª. Simone Nasser Matos

PONTA GROSSA

2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Ponta Grossa
Diretoria de Pesquisa e Pós-Graduação
Departamento Acadêmico de Informática
Bacharelado em Ciência da Computação

TERMO DE APROVAÇÃO

APLICAÇÃO DE BIOINFORMÁTICA PARA RECONHECIMENTO DE PLÁGIO EM CÓDIGOS DE PROGRAMAÇÃO

Por

KAIO PABLO GOMES

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 6 de novembro de 2017 como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação. O candidato foi arguido pela Banca Examinadora composta pelos professores abaixo assinados. Após deliberação, a Banca Examinadora considerou o trabalho aprovado.

____
Prof. Dra. Simone Nasser Matos
Prof.(a) Orientador(a)

____
Prof. Dr. André Pinz Borges
Membro titular

____
Profa. Dra. Simone de Almeida
Membro titular

____
Profª. Dra. Helyane Bronoski Borges
Responsável pelo Trabalho de Conclusão de Curso

____
Prof. MSc. Saulo Jorge Beltrão de Queiroz
Coordenador do curso

- O Termo de Aprovação assinado encontra-se na Coordenação do Curso

AGRADECIMENTOS

A realização deste trabalho foi possível graças a participação da professora Simone Nasser Matos, que sempre me apoiou durante toda a jornada acadêmica. Tendo um papel fundamental de participação por meio de correções e orientações durante o trabalho de conclusão de curso.

A professora Kate Cooper por me introduzir na área de bioinformática e possibilitou a minha participação em uma pesquisa, na qual foi motivante para o desenvolvimento do tema deste trabalho.

Ao pesquisador Jay Pedersen, por me orientar durante uma pesquisa realizada na Universidade de Nebraska em Omaha. Jay Pedersen me forneceu todo o suporte necessário para o desenvolvimento deste trabalho.

Aos meus pais e amigos pelo apoio incondicional em todos os momentos durante toda a jornada acadêmica. Agradeço a Deus por tudo.

RESUMO

GOMES, Kaio Pablo. **Aplicação de Bioinformática para o reconhecimento de plágio em códigos de programação**. 2017. 66 f. Trabalho de Conclusão de Curso de Bacharelado em Ciência da Computação - Universidade Tecnológica Federal do Paraná. Ponta Grossa, 2017.

A bioinformática é um estudo interdisciplinar que busca a solução para problemas de diferentes áreas da ciência tais como: biologia molecular, biologia celular, bioquímica, química, física e computação. Para a computação, a bioinformática pode resolver problemas tal como a detecção de vírus de computador, usando uma metodologia que se baseia em arquivo código. Esta metodologia pode ser usada na detecção de plágio em código de programação, pois estes possuem também um arquivo binário. Este trabalho realizou a aplicação desta metodologia no domínio de reconhecimento de plágio em códigos de programação criando um conjunto de teste para mostrar a sua viabilidade. Os resultados da aplicação dos testes mostram o potencial da metodologia, porém dificuldades em identificar alguns tipos de plágios foram encontradas e melhorias tais como: utilização de filtros de normalização de códigos de programação e mudança no modelo de alinhamento de sequências foram apresentadas para superar eventuais desafios.

Palavras-chave: Bioinformática. Metodologia. Plágio. Códigos de programação. Arquivo Binário.

ABSTRACT

GOMES, Kaio Pablo. **Application of Bioinformatics for the plagiarism recognition in programming codes.** 2017. 66 p. Dissertation (Bachelor of Computer Science) – Federal Technological University of Paraná. Ponta Grossa, 2017.

The bioinformatics is an interdisciplinary study that seeks the solution for the most diverse problems of different science fields such as molecular biology, cellular biology, biochemistry, chemistry, physics and computing. For computing, the bioinformatics can solve problems such as the computer virus detection, by using a methodology that is based on binary file. This methodology can be used in the detection of plagiarism in programming code, since these also have a binary file. This project performed the application of this methodology in the field of plagiarism recognition in programming codes creating a test set to show its viability. The application results of the tests show the potential of the methodology, but some difficulties in identifying some types of plagiarism were found and improvements such as: use of programming code normalization filters and change in the sequence alignment model were presented to overcome eventual challenges.

Key-words: Bioinformatics. Methodology. Plagiarism. Programming codes. Binary file.

LISTA DE FIGURAS

FIGURA 1 – ÁREAS DE ESTUDO DA BIOINFORMÁTICA 18
FIGURA 2 – FUNCIONAMENTO DO ALINHAMENTO DE DNA 20
FIGURA 3 – RESULTADO DE UM ALINHAMENTO 21
FIGURA 4 - ALINHAMENTO LOCAL ENTRE DUAS SEQUÊNCIAS 22
FIGURA 5 - PLÁGIO DE CÓDIGO TIPO I 29
FIGURA 6 - PLÁGIO DE CÓDIGO TIPO II 30
FIGURA 7 - PLÁGIO DE CÓDIGO TIPO III 31
FIGURA 8 - PLÁGIO DE CÓDIGO TIPO IV 32
FIGURA 9 - NORMALIZAÇÃO COM O ALGORITMO *SHERLOCK* 34
FIGURA 10 - CÓDIGO DE PROGRAMAÇÃO EM AST 36
FIGURA 11 - CÓDIGO DE PROGRAMAÇÃO EM UM GRAFO 37
FIGURA 12 - FUNCIONAMENTO DA FERRAMENTA *JPLAG* 39
FIGURA 13 - FUNCIONAMENTO DA FERRAMENTA MOSS 40
FIGURA 14 - PROCESSO DE APLICAÇÃO DA METODOLOGIA 41
FIGURA 15 - MAPEAMENTO DE BITS EM BASES 43
FIGURA 16 - TRANSFORMAÇÃO EM BASES 44
FIGURA 17 – EMBOSS ALINHAMENTO DE DNA SINTÉTICO 45
FIGURA 18 – RELATÓRIO BLAST DE ALINHAMENTO DE DNA SINTÉTICO 46
FIGURA 19 – RESULTADOS OBTIDOS 48
FIGURA 20 - PSEUDOCÓDIGO DA PROPOSTA DE MUDANÇAS 53
FIGURA 21 – CONJUNTO DE TESTE A, CÓDIGO 1 62
FIGURA 22 - CONJUNTO DE TESTE A, CÓDIGO 2 63
FIGURA 23 - CONJUNTO DE TESTE A, RESULTADO 63
FIGURA 24 - CONJUNTO DE TESTE B, CÓDIGO 1 64
FIGURA 25 - CONJUNTO DE TESTE B, CÓDIGO 2 64
FIGURA 26 - CONJUNTO DE TESTE B, RESULTADO 64
FIGURA 27 - CONJUNTO DE TESTE C, CÓDIGO 1 65
FIGURA 28 - CONJUNTO DE TESTE C, CÓDIGO 2 65
FIGURA 29 - CONJUNTO DE TESTE C, RESULTADO 66

SUMÁRIO

1 INTRODUÇÃO 13
1.1 OBJETIVOS 14
1.2 ORGANIZAÇÃO DO TRABALHO 15
2 BIOINFORMÁTICA: UMA VISÃO GERAL 17
2.1 BIOINFORMÁTICA 17
2.2 MODELAGEM E APLICAÇÕES DE DNA SINTÉTICO 19
2.3 DEFINIÇÕES DE DNA E ALINHAMENTO 19
2.4 TIPOS DE ALINHAMENTO DE DNA 22
2.5 FERRAMENTAS DE ALINHAMENTO DE DNA 23
2.5.1 Ferramenta BLAST 23
2.5.2 Ferramenta EMBOSS 23
2.6 ANÁLISE QUALITATIVA DE FERRAMENTAS DE DNA 24
2.7 TRABALHOS RELACIONADOS 26
3 IDENTIFICAÇÃO DE PLÁGIO 28
3.1 DEFINIÇÕES DE PLÁGIO 28
3.2 TIPOS DE PLÁGIO 29
3.3 TÉCNICAS DE IDENTIFICAÇÃO DE PLÁGIO 32
3.3.1 Técnica Baseada em Texto 32
3.3.2 TÉCNICA BASEADA EM TOKEN 34
3.3.3 Técnica Baseada em Árvore 35
3.3.4 Técnica Baseada em Grafos 36
3.3.5 Técnica Baseada em Métricas 37
3.4 FERRAMENTAS DE PLÁGIO 38
4 APLICAÇÃO DA BIOINFORMÁTICA EM DETECÇÃO DE PLÁGIOS 41
4.1 PROCESSO GERAL DE APLICAÇÃO DO ALINHAMENTO 41
4.2 ARTEFATO COMPUTACIONAL 42
4.3 ALGORITMO GERADOR DE DNA SINTÉTICO 42
4.4 FERRAMENTA PARA ALINHAMENTO DE DNA 44
4.5 RELATÓRIO DO ALINHAMENTO DE DNA 45
5 RESULTADOS 48
5.1 CONJUNTO DE TESTES 48
5.2 ANÁLISE DOS CONJUNTOS DE TESTES 50
5.3 PROPOSTAS DE MUDANÇAS NA METODOLOGIA BASEADA EM BIOINFORMÁTICA 52
6 CONCLUSÃO 56
6.1 TRABALHOS FUTUROS 57
REFERÊNCIAS 59
APÊNDICE A – CONJUNTO DE TESTES ADICIONAIS 62

1 INTRODUÇÃO

O plágio no meio acadêmico tem se tornado uma preocupação para as instituições de ensino. Conforme PLAG.PT (2016), uma pesquisa realizada em 2014 nos Estados Unidos, mostra que as taxas de plágio no ambiente acadêmico americano chegaram a atingir 21,8% como no caso do estado do Texas.

A detecção de plágio em códigos de programação é uma área que possui diversos estudos que usam uma determinada técnica de modelagem do problema, sendo uma das principais o algoritmo de *Sherlock*. Esta técnica busca desconsiderar previamente informações desnecessárias em códigos de programação tais como espaçamento, comentários e nomes de variáveis antes de realizar uma comparação de similaridade com o intuito de procurar plágio (MACIEL *et al*, 2012).

Uma técnica alternativa para o problema de identificação de plágio em códigos de programação é o alinhamento de DNA (Ácido Desoxirribonucleico). O DNA é uma cadeia polinucleotídica que pode ser representada por uma sequência de nucleotídeos (PEDERSEN, 2012). O alinhamento de DNA se tornou um campo de estudo importante para a bioinformática, tornando possível a criação de relacionamentos evolutivos de espécies assim como a identificação de funções moleculares de proteínas (BRITO, 2003).

A bioinformática aproveita os estudos de alinhamento de DNA para modelar e gerar soluções para problemas computacionais. Estudos realizados sugerem que algumas das principais aplicações de alinhamento de DNA podem ser em: identificação de *malwares* de computadores, plágio em códigos de programação e processamento de formato de arquivos (PEDERSEN, 2012). A aplicação na área de identificação de plágio demonstra a possibilidade de obter melhores resultados e em menor tempo possível em relação a outras técnicas de identificação de plágio.

A identificação de plágio em códigos de programação é uma aplicação que pode ser modelada dentro do campo de estudo de alinhamento de DNA, pois depende da criação de um modelo representativo dos códigos de programação. Para esta representação é possível abstrair a ideia de um modelo conhecido como DNA sintético.

O DNA sintético é baseado na estrutura de um DNA e é feita a partir de uma sequência de bases nitrogenadas conhecidas como Adenina, Guanina, Timina e

Citosina (HUNTER, 2009). Por outro lado, todo e qualquer artefato computacional é estruturado por meio de uma sequência binária.

A criação de um DNA sintético pode ser feita a partir de um mapeamento de uma sequência binária em uma sequência de bases nitrogenadas. Por exemplo, o código binário 00 representa A, código 11 representa G, código 01 representa T e código 10 representa C. As abreviações das bases nitrogenadas são conhecidas como: A, T, C e G. Desta forma, é possível representar um artefato computacional, neste caso, os códigos de programação em um modelo simplificado de DNA, conhecido como DNA sintético (PEDERSEN, 2012).

Após a representação de um DNA sintético, a próxima etapa é do alinhamento de DNA que consiste em comparar as sequências de DNA sintéticos previamente criados. A comparação é realizada por meio da concepção de alinhamento e conta com alguns critérios específicos de pontuação para definir o nível de similaridade entre as sequências comparadas (VIANA, 2010).

Com a utilização de ferramentas de bioinformática específicas de alinhamento de sequências, um relatório de alinhamento é gerado e desta forma utilizado para identificar a taxa de similaridade encontrada entre duas ou mais sequências de DNA.

Este trabalho aplicou a metodologia de bioinformática de Pedersen (2012) no reconhecimento de plágio em códigos de programação. Um conjunto de testes foram aplicados para identificar a viabilidade da metodologia em atuar no campo da computação de identificação de plágio. Usou uma ferramenta para realizar o alinhamento de sequência e por meio do relatório gerado considerou os parâmetros para análise, a saber, a taxa de identidade e similaridade, para definir uma porcentagem de similaridade entre os DNA sintéticos, que neste trabalho são os códigos de programação.

A partir da análise dos resultados de testes aplicados, alguns aprimoramentos e ajustes na metodologia utilizada são sugeridos.

1.1 OBJETIVOS

O objetivo geral é aplicar a metodologia de Pedersen (2012) para mostrar a viabilidade de utilização da bioinformática no reconhecimento de plágio em códigos de programação.

Os objetivos específicos são:

- Modelar artefatos computacionais, neste caso, códigos de programação.
- Identificar padrões de resultados de alinhamentos a fim de garantir uma identificação de plágio em códigos de computadores de modo a contemplar os tipos de plágios dessa área.
- Propor modificações na metodologia de Pederson (2012) a fim de obter melhores resultados na aplicação da bioinformática no domínio de plágio de código de programação.

1.2 ORGANIZAÇÃO DO TRABALHO

A organização deste trabalho consiste de seis capítulos. O capítulo 2 introduz a importância da bioinformática em soluções de problemas computacionais e conceitos relacionados ao processo de alinhamento de DNA. Também apresenta como a bioinformática possibilita a modelagem de artefatos computacionais em DNA sintético assim como suas possíveis aplicações. Ao final deste capítulo, um estudo sobre os processos de alinhamento de DNA é apresentado e exemplifica como é feito o alinhamento de um DNA sintético.

O capítulo 3 apresenta os principais conceitos relacionados à identificação de plágio. Uma discussão das técnicas de identificação de plágio em códigos disponíveis na computação e a realização de uma análise das principais ferramentas e a metodologia utilizada por cada uma das ferramentas analisadas são relatadas.

O capítulo 4 descreve a metodologia baseada em bioinformática para reconhecimento de plágio em códigos de programação usada por este trabalho. O capítulo inicialmente apresenta o desenvolvimento de um algoritmo de geração de DNA sintético e descreve a integração do mesmo com a ferramenta de alinhamento de sequências. Um conjunto de testes é desenvolvido para a demonstração da aplicação deste método.

O capítulo 5 apresenta a aplicação da metodologia por meio de um conjunto de testes. O conjunto de testes foi elaborado para avaliar a metodologia na identificação de plágios. Uma análise dos resultados obtidos, assim como propostas de mudanças na metodologia foram apresentadas ao final do capítulo.

No último capítulo, o capítulo 6, mostra a conclusão dos resultados alcançados por meio da aplicação do conjunto de testes. De uma forma geral os aspectos relacionados a metodologia de bioinformática são apresentados, de modo, a mostrar a sua viabilidade na área de identificação de plágio em códigos de programação.

2 BIOINFORMÁTICA: UMA VISÃO GERAL

Este capítulo apresenta como a bioinformática está relacionada com o tema desta pesquisa de modo a explicar a ideia de um DNA sintético. A seção 2.1 mostra uma breve descrição das áreas de estudos da bioinformática assim como a área da bioinformática que foi estudada para o desenvolvimento deste trabalho. A seção 2.2 relata a ideia de um DNA sintético por meio de modelagem de artefatos computacionais e apresenta as aplicações que podem ser estudadas com a utilização de bioinfomática para soluções de problemas computacionais utilizando DNA sintético. A seção 2.3 apresenta os conceitos e definições de uma sequência de DNA e o processo de alinhamento de sequências de DNA. A seção 2.4 mostra os diferentes tipos de alinhamento de sequências. A seção 2.5 apresenta um estudo das principais ferramentas de bioinformática para realizar alinhamento de sequências de DNA. A seção 2.6 realiza uma análise qualitativa das ferramentas de alinhamento de sequências de DNA estudadas na seção anterior. Por fim, a seção 2.7 mostra os principais trabalhos relacionados com a metodologia estudada nesse trabalho.

2.1 BIOINFORMÁTICA

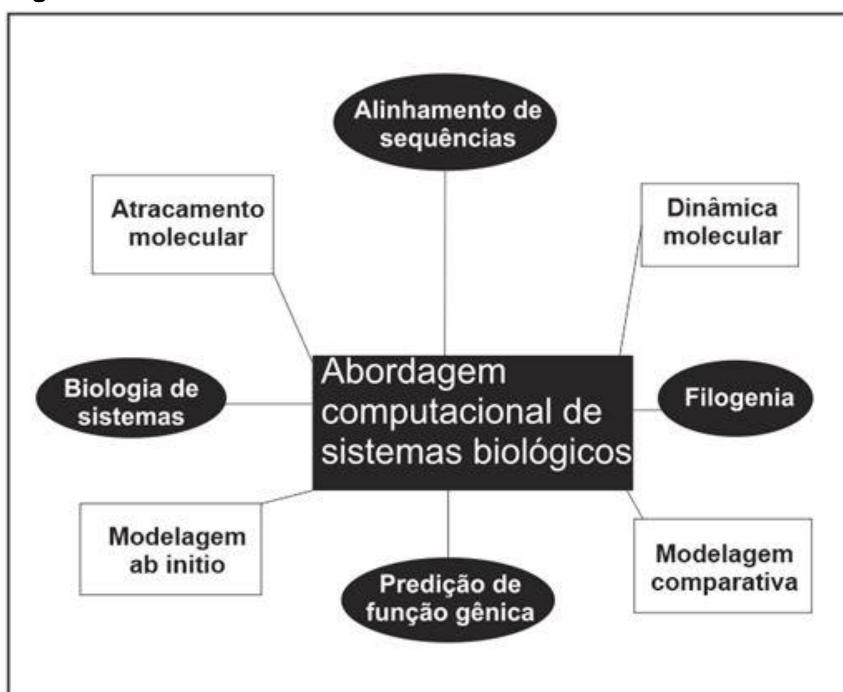
A bioinformática é uma área recente da ciência e seus estudos são caracterizados por serem interdisciplinares de modo a envolver diversas áreas do conhecimento tais como: biologia molecular, biologia celular, bioquímica, química, física e computação. A bioinformática busca resolver problemas relacionados à estrutura de biomoléculas e sequências de biomoléculas. As principais biomoléculas estudadas são ácidos nucleicos e proteínas (WATERMAN, 1995).

De uma maneira resumida, a figura 1, representa os diversos campos de estudos da bioinformática baseados nos dois campos centrais: de estrutura e sequências de biomoléculas. Os campos relacionados à estrutura de biomoléculas estão representados pelas formas geométricas retangulares e os campos relacionados a sequências de biomoléculas estão representados pela forma geométrica circular (VERLI, 2014).

O foco de estudo da área de estrutura de biomoléculas é associado ao entendimento das moléculas e situações desencadeadas pela ação de moléculas. Alguns exemplos de campos de estudos são: Identificação do modo de interação de moléculas e avaliação dos efeitos das mudanças na estrutura e ambiente molecular. Os seus campos de pesquisas são: atracamento molecular, dinâmica molecular, modelagem comparativa e modelagem *ab initio*.

Por outro lado, o estudo das sequências de biomoléculas possui como foco principal a manipulação de sequências de genomas inteiros que contribui para um entendimento dos organismos. Alguns exemplos de estudos desta área são: comparação entre sequências, identificação de padrões em sequências e avaliação de relações evolutivas entre organismos. As áreas de estudos são: Alinhamento de sequências, filogenia, biologia de sistemas e predição de função gênica (VERLI, 2014).

Figura 1 – Áreas de estudo da bioinformática



Fonte: Adaptado de Verli (2014, p. 17)

Este projeto de pesquisa trabalha com os conceitos e aplicações envolvidos em um dos campos de estudo de sequências de biomoléculas. A metodologia, baseada em bioinformática, abordada neste trabalho está vinculada ao campo de alinhamento de sequências.

2.2 MODELAGEM E APLICAÇÕES DE DNA SINTÉTICO

Um artefato computacional é considerado todo e qualquer tipo de arquivo tal como: foto digital, arquivo de texto, código-fonte de programação dentre outros (PEDERSEN, 2012). A ideia de modelar artefatos computacionais em sequências de DNA está vinculada com a possibilidade de utilizar ferramentas de bioinformática. Neste trabalho, os artefatos computacionais trabalhados são os códigos fontes de programação.

A estruturação de um DNA (Ácido Desoxirribonucleico) é feita a partir de uma sequência de bases nitrogenadas conhecidas como Adenina, Guanina, Timina e Citosina (HUNTER, 2009). Por outro lado, todo e qualquer artefato computacional é estruturado por meio de uma sequência binária. O conceito de DNA sintético está associado a um modelo de DNA em que é composto do mapeamento de uma sequência binária em uma sequência de bases nitrogenadas (PEDERSEN, 2012).

Ao modelar artefatos computacionais em sequências de DNA se torna possível a aplicação de ferramentas de bioinformática. A utilização de ferramentas de alinhamento de sequências pode ser aplicada em diferentes campos de estudo da computação. Alguns dos principais campos de estudo são: segurança da informação e identificação de plágio.

O campo de segurança da informação pode definir novos métodos de identificação de *malwares* de modo a identificar arquivos infectados. Para o campo de plágio a possibilidade de identificar o nível de similaridade entre arquivos. Outras aplicações como reconhecimento de tipos de arquivos podem ser consideradas para estudo (PEDERSEN, 2012).

2.3 DEFINIÇÕES DE DNA E ALINHAMENTO

O ácido desoxirribonucleico (DNA) é responsável por armazenar informações genéticas de um organismo. A sua estrutura é composta de duas cadeias longas polinucleotídicas formada por quatro tipos de unidades menores conhecidas como nucleotídeos. Os nucleotideos são formados por três moléculas: um fosfato, uma pentose e uma base nitrogenada. O funcionamento do DNA depende de um processo de codificação da informação genética do indivíduo. O

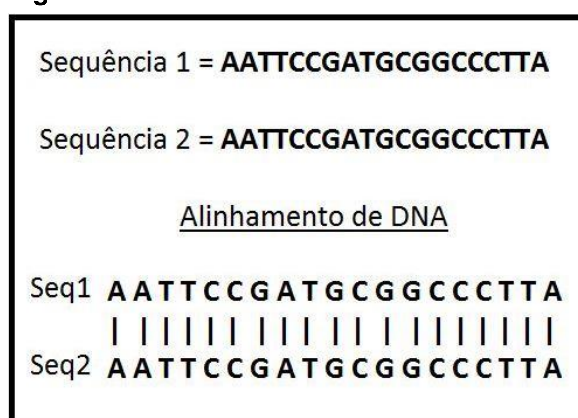
processo de codificação da informação genética é representado pela sequenciação das bases nitrogenadas das estruturas polinucleotídicas. As bases nitrogenadas são timina (*t*), adenina (*a*), citosina (*c*) e guanina (*g*) (ZAHA apud NETO, 2008).

A representação de uma sequência de DNA para realizar um alinhamento é comumente definida por um alfabeto ∑. O alfabeto é um conjunto finito de símbolos não nulos representados como ∑ = {A,C,G,T,-}. Os seguintes simbolos: A,C,G e T são referências as bases nitrogendas de uma estrutura de DNA. O simbolo “-“ é uma indicação de espaço em branco (VIANA, 2010).

O alinhamento de DNA pode ser compreendido como uma maneira de organizar duas ou mais sequências de bases nitrogenadas. Esta organização de sequências tem como objetivo realizar a identificação de similaridades entre as sequências envolvidas. Estas regiões de similaridade indicam algum tipo de relacionamento entre os organismos representados pelas sequências (WATERMAN, 1995).

Com uma representação definida para as sequências, o processo de alinhamento se torna possível. Existem diversas ferramentas computacionais que auxiliam o processo de alinhamento de sequências tanto para o domínio de DNA quanto outros domínios tais como RNA e proteinas. A figura 2 representa um esboço do alinhamento entre duas sequências de DNA.

Figura 2 – Funcionamento do alinhamento de DNA



Fonte: Autoria própria

O resultado de um alinhamento de sequências é um valor determinado a partir de critérios de pontuações. Os critérios têm como objetivo determinar o nível de similaridade entre duas ou mais sequências analisadas no processo de alinhamento. Os critérios são definidos por três tipos principais: igualdade (*match*),

desigualdade (*mismatch*) e espaçamento (*gap*). O critério de igualdade é definido para os casos de dois símbolos comparados serem iguais. O critério de desigualdade representa casos de dois símbolos comparados serem diferentes. Para o critério de espaçamento a representação realizada é de casos de comparação de um símbolo com um espaço em branco ou entre dois espaços em brancos.

Uma quantidade significativa de algoritmos de alinhamento utiliza uma pontuação padrão para os casos de igualdade, desigualdade e espaçamento. A pontuação comumente empregada é a seguinte: atribui-se um ponto para casos de igualdade, retira-se um ponto para casos de desigualdade e retira-se dois pontos para caso de espaçamento. O resultado de um alinhamento é a soma de todas as pontuações parciais obtidas em cada caso. Quanto maior este valor do resultado, maior o nível de similaridade (VIANA, 2010).

Com o intuito de demonstrar o funcionamento de um alinhamento de DNA, a figura 3 ilustra esboço do processo de alinhamento de duas sequências de DNA com base nos critérios de igualdade, desigualdade e espaçamento.

Figura 3 – Resultado de um alinhamento

Sequência 1:	A	T	C	G	T	A
Sequência2:	A	T	A	C	C	-
total:	+1	+1	-1	-1	-1	-2

TOTAL = +1+1-1-1-1-2 = -3

Fonte: Autoria própria

Nota-se neste exemplo da figura 3, um resultado gerado do alinhamento de duas sequências. Este resultado denominado “TOTAL” é calculado por meio do somatório de pontuações individuais para cada comparação de símbolos. Desta forma, as pontuações individuais correspondem as bases nitrogenadas alinhadas. Por exemplo, inicialmente a base “A” da sequência 1 foi alinhada com a base “A” da sequência 2, de forma a gerar uma pontuação +1 devido as bases serem iguais. No último alinhamento, a sequência 1 possui a base “A” enquanto que a sequência 2 possui um espaço, ou seja, não possui bases naquela região. Com isso, é um caso

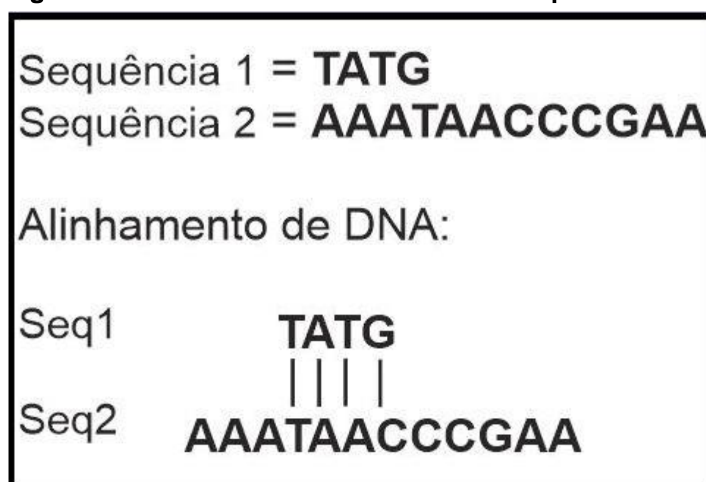
típico de espaçamento no qual resulta a pontuação -2. Assim é calculado para cada pontuação individual respeitando o critério discutido anteriormente neste capítulo.

2.4 TIPOS DE ALINHAMENTO DE DNA

O alinhamento de sequência é um procedimento que depende da forma como são organizadas as sequências de DNA, RNA ou de proteína (WATERMAN apud VIANA; 2010). Na literatura de alinhamentos em bioinformática, classifica-se o processo de alinhamento em duas categorias: alinhamento global e o local.

O alinhamento global é um tipo de alinhamento em que as sequências são comparadas de uma extremidade até a outra. As regiões com espaço em branco são representadas para cada sequência caso exista diferença de tamanho entre duas sequências. As figuras 2 e 3 representam este tipo de alinhamento, enquanto que a figura 4 representa um esboço do alinhamento local (VIANA, 2010).

Figura 4 - Alinhamento local entre duas sequências de DNA



Fonte: Autoria própria

O alinhamento local, ilustrado na figura 4, é um tipo de alinhamento em que se procura o melhor posicionamento do alinhamento com base nos critérios de semelhanças encontrados. Este tipo de alinhamento pode ser usado de maneira a não definir uma região específica e gerar como resultado a identificação de uma região que apresenta alta taxa de similaridade (VIANA, 2010).

2.5 FERRAMENTAS DE ALINHAMENTO DE DNA

O processo de alinhamento de DNA é realizado por meio de algoritmos de alinhamento de sequências. A seguir nas seções 2.5.1 e 2.5.2 duas ferramentas de bioinformática para alinhamento de sequências são apresentadas. Estas são duas das ferramentas mais utilizadas de bioinformática.

2.5.1 Ferramenta BLAST

A ferramenta *Basic Local Alignment Search Tool* (BLAST) é utilizada para realizar alinhamentos do tipo local. O seu funcionamento é compatível com diferentes tipos de sequências. Pode-se fazer alinhamento local para sequências de proteina, ácido ribonucleico (RNA) ou DNA (BLAST, 2016).

A ferramenta BLAST é de código aberto, com isso diversas implementações desta ferramenta foram feitas. As mais conhecidas são NCBI-BLAST desenvolvida pela *National Center for Biotechnology Information* e WU-BLAST implementada pela Universidade de Washington (SANTOS, 2004).

A NCBI-BLAST possui diversos bancos de dados genéticos bem como sequências de DNA de diversos organismos (NCBI, 2016). Por esta razão, é a ferramenta mais utilizada entre os pesquisadores e estudiosos de áreas de conhecimentos correlacionados a bioinformática.

A NCBI classifica a ferramenta BLAST em cinco tipos: *blastp* responsável por fazer alinhamento de aminoácidos, *blastn* tem como objetivo fazer o alinhamento geral de sequências de nucleotídeos, blastx responsável por fazer alinhamento de nucleotídeo transladado, *tblastn* realiza a comparação de sequências de proteinas em relação a nucleotídeos e *tblastx* faz o alinhamento específico de nucleotídeos (HIGA apud SANTOS, 2004).

2.5.2 Ferramenta EMBOSS

A ferramenta EMBOSS é utilizada para realizar alinhamento do tipo global entre as sequências. Esta ferramenta é mantida pela *European Bioinformatics*

Institute (EMBL). O seu funcionamento é compátivel com sequências de proteinas, RNA ou DNA (SANTOS, 2004).

A licença da ferramenta EMBOSS é de código aberto. A sua utilização é flexível. Os domínios das sequências não precisam ser identificados. Com isso, o procedimento para realizar alinhamento é igual para sequências de proteinas, RNA ou DNA.

2.6 ANÁLISE QUALITATIVA DE FERRAMENTAS DE DNA

A análise comparativa de ordem qualitativa foi realizada para as ferramentas: EMBOSS e BLAST. Ambas as ferramentas realizam alinhamento de sequências de DNA e na área de bioinformática são populares entre os pesquisadores e estudiosos. Os critérios de análise para estas ferramentas são os seguintes:

- Tipo de licença de software: Apresentação do tipo de licença em relação ao seu uso não comercial. Os critérios são: (*) para código aberto e (**) para software livre. O critério código aberto está associado as ferramentas que podem ser disponibilizadas e estudadas livremente, porém com a restrição de autorização dos autores para que possa ser implementada variações da ferramenta. O critério software livre é para as ferramentas totalmente disponibilizadas ao público sem precisar inclusive de autorização para implementar variações desta ferramenta. Este critério de licença de software é considerado importante uma vez que haja a possibilidade da comunidade cientifica estudar e inclusive implementar ferramentas de alinhamento com base em modelos consolidados já existentes.
- Plataforma *Web Based* (baseado na internet): A existência ou não de versão web das ferramentas análisadas. Os critérios são: sim ou não.
- Manual: A presença de algum documento oficial de caráter explicativo para o funcionamento da ferramenta. Os critérios são: sim ou não.
- Flexibilidade: A questão de flexibilidade está associada com a possibilidade de fazer o alinhamento de sequências sem precisar informar o tipo de sequência. Os critérios são: presente ou ausente.
- Alinhamento de proteínas: Se a ferramenta realiza alinhamento de

sequências de proteínas. Os critérios são: sim ou não.

- Alinhamento de DNA: Se a ferramenta realiza alinhamento de sequências de DNA. Os critérios são: sim ou não.
- Alinhamento de RNA: Se a ferramenta realiza alinhamento de sequências de RNA. Os critérios são: sim ou não.
- Tipo de alinhamento: Determinar qual o tipo de alinhamento realizado. Os critérios adotados são: local e global. Base de dados própria Integrada: Verificar se a ferramenta fornece algum tipo de complemento a fim de possibilitar ao usuário a criação de um próprio banco de dados de sequências integrado a ferramenta. Os critérios são: ausente e presente.
- FAQ: Indicar se a ferramenta possui um sistema de ajuda *Frequently Asked Questions* (FAQ). Os critérios são: sim ou não.
- Solução ótima: Verificar se ao executar o alinhamento de sequências, a ferramenta encontra uma solução ótima. Os critérios são sim ou não.

O resumo da análise comparativa de ordem qualitativa é apresentado no quadro 1. De modo a relacionar cada item de critério avaliativo com uma determinada ferramenta.

Quadro 1 - Resumo da análise qualitativa aplicada

Critérios adotados	**Ferramenta analisada**	
	BLAST	**EMBOSS**
Tipo de licença de software	**	*
Plataforma Web Based	Sim	Sim
Manual	Sim	Sim
Flexibilidade	Ausente	Presente
Alinhamento de proteínas	Sim	Sim
Alinhamento de DNA	Sim	Sim
Alinhamento de RNA	Sim	Sim
Tipo de alinhamento	Local	Global
Base de dados Própria Integrada	Presente	Ausente
FAQ	Sim	Sim

Fonte: Autoria própria

De acordo com o quadro 1, as ferramentas *Blast* e *EMBOSS* apresentam

diversas similaridades, tais como: Plaforma *Web Based*, manual, alinhamento de proteínas, alinhamento de DNA, tipo de alinhamento e FAQ. Ambas as ferramentas não possuem um nível de usabilidade tão satisfatório. Este é um dos principais problemas encontrados na maioria das ferramentas de alinhamento de sequências disponíveis, tendo em vista que as duas ferramentas analisadas são as mais populares.

O principal critério de diferença da ferramenta BLAST é a presença de base de dados própria integrada, uma vez que esta funcionalidade é desejável para a comunidade de pesquisadores da área. Para a ferramenta EMBOSS, o seu principal diferencial é a flexibilidade de trabalhar com sequências sem se preocupar com seu tipo. Esta flexibilidade é interessante em atividades regulares de realização de alinhamento. Observa-se que ambas as ferramentas apresentam possiblidades de executar os dois tipos de alinhamento.

2.7 TRABALHOS RELACIONADOS

Uma metodologia de reconhecimento de plágio em códigos-fontes baseada em alinhamento de sequências e elementos de uma árvore sintática abstrata foi proposta por Kikuchi *et al.* (2014). O trabalho busca criar uma ferramenta de identificação de plágio precisa que consegue garantir a percepção de técnicas de mudanças de variáveis, nomes de funções e ordem das linhas de códigos-fontes.

O funcionamento da metodologia é baseado em algoritmos de alinhamento de sequência como o *Needleman-Wunsch algorithm,* que utiliza programação dinâmica para realizar alinhamento global de duas ou mais sequências. Este algoritmo é utilizado nas principais ferramentas de bioinformática, como por exemplo, o EMBOSS já o utiliza para realizar alinhamentos do tipo global. Outro aspecto importante é a utilização da técnica de árvores sintáticas abstratas, que conta com a geração de *tokens* por meio de um analisador léxico.

A metodologia recebe como entrada dois códigos-fontes, os mesmos são analisados do ponto de vista léxico. Após o analisador léxico gerar *tokens*, duas sequências de *tokens* são criadas representando, cada uma, um código de programação de entrada. Para uma das sequências é feita uma separação por unidades de funções. A aplicação de um separador em uma das sequências é feita

de modo a transformar os conjuntos de *tokens* em unidades de funções.

Com a sequência de *tokens* de um código de programação de entrada e as unidades de funções geradas a partir do outro código de programação de entrada, realiza-se o alinhamento de sequências do tipo global para todas as combinações possíveis entre uma sequência de *tokens* e as unidades de funções.

Durante o processo de alinhamento, identifica-se o alinhamento com maior pontuação e o classifica como sendo a pontuação máxima. Os resultados de alinhamentos das outras combinações são analisados para tirar uma média. O grau de similaridade será a média de pontuação normalizada com a pontuação máxima.

Uma outra metodologia de bioinformática proposta por Pedersen (2012), na qual baseia-se em alinhamento de DNA no reconhecimento de vírus de computador utilizando a ferramenta BLAST, possui possibilidade de aplicação na área de reconhecimento de plágio em códigos de programação.

Diferente da metodologia de KIKUCHI *et al* (2014), a proposta de Pedersen *et al* (2012) de identificar plágio em códigos de programação é baseada no acesso binário dos códigos analisados. Desta forma, busca eliminar as possibilidades de camuflagem existentes no processo de plágio. Porém, ambas as metodologias se baseiam em alinhamento de sequências.

Neste trabalho, é feita a aplicação da metodologia de Pedersen (2012) na área de plágio de códigos de programação devido a sua peculiaridade de acesso ao binário de um código de programação. Com esse diferencial de acesso ao binário aliado a propostas de melhorias, uma nova metodologia aprimorada contempla a proposta desse trabalho.

3 IDENTIFICAÇÃO DE PLÁGIO

Este capítulo apresenta os conceitos relacionados ao plágio de modo a apresentar as definições, tipos, técnicas e ferramentas. A seção 3.1 descreve as principais definições de plágio. A seção 3.2 relata os tipos de plágio. A seção 3.3 mostra as técnicas de identificação de plágio. Por fim, na seção 3.4 são apresentadas duas ferramentas de identificação de plágio mais usadas.

3.1 DEFINIÇÕES DE PLÁGIO

Conforme o dicionário Cambridge (2016), o plágio pode ser definido como "Usar o trabalho ou ideia de outra pessoa e fingir que é o seu próprio trabalho ou ideia". Existem outras definições de plágio. Segundo Merrian-Webster (2016) o plágio pode ser:

- Roubar e passar (as ideias ou palavras de outro) como se fossem suas.
- Usar (produção de outro) sem creditar a fonte.
- Cometer literalmente roubo.
- Apresentar como nova e original uma ideia ou produto derivado de uma fonte existente.

Para o contexto de plágio em ciência da computação, a programação está bastante associada com o plágio. O plágio associado a programação é caracterizado como sendo plágio de código de programação. O que caracteriza esta forma de plágio é: (COSMA; JOY apud LIMA, 2010):

- Reutilização de código-fonte sem fornecer referência adequada;
- Conversão integral ou parcial para outra linguagem de programação;
- Usar *software* para gerar o código-fonte sem alertar o fato;
- Pagar alguém para fazer.

O plágio tem se tornado um grande problema para as instituições de ensino. No ano de 2014, segundo PLAG.PT (2016) um levantamento estatístico de plágio na Europa e nos Estados Unidos identificou a necessidade de controlar o plágio nas instituições de ensino. Os dados na Europa mostram que as taxas de plágio variam

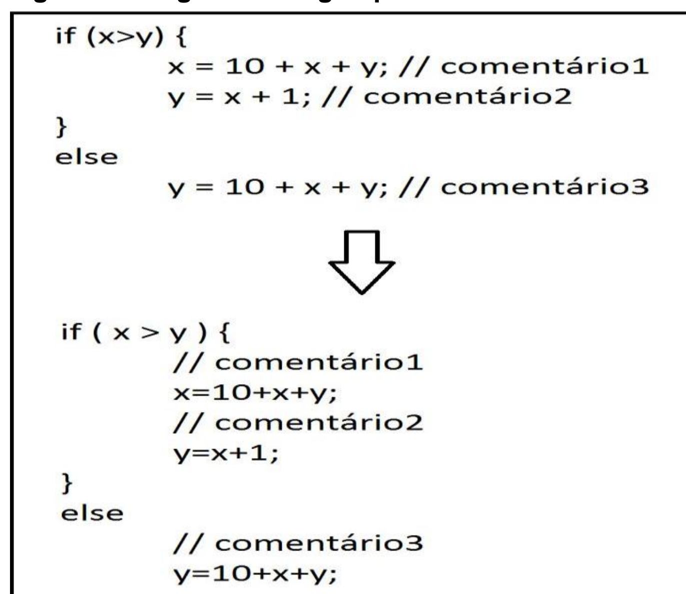
de 4,7% até 26,1%. Enquanto que nos Estados Unidos a mesma pesquisa mostra uma taxa de plágio variando de 6,4% até 24.2%.

3.2 TIPOS DE PLÁGIO

Considerando o contexto de plágio em códigos de programação, a literatura define como *clone* de código os fragmentos de códigos idênticos ou similares. A partir desta definição é possível determinar os tipos de clonagem de códigos. Existem três tipos principais de clonagem de códigos: Tipos I, II, III e IV (ROY; CORDY, 2007).

O Tipo I é baseado em fragmentos de códigos idênticos, mas com variações em espaçamento e comentários (ROY; CORDY, 2007). A figura 5 ilustra um esboço deste tipo de plágio: Um fragmento de código é copiado, mas com algumas modificações em relação ao espaçamento do código e a localização dos comentários.

Figura 5 - Plágio de código tipo I

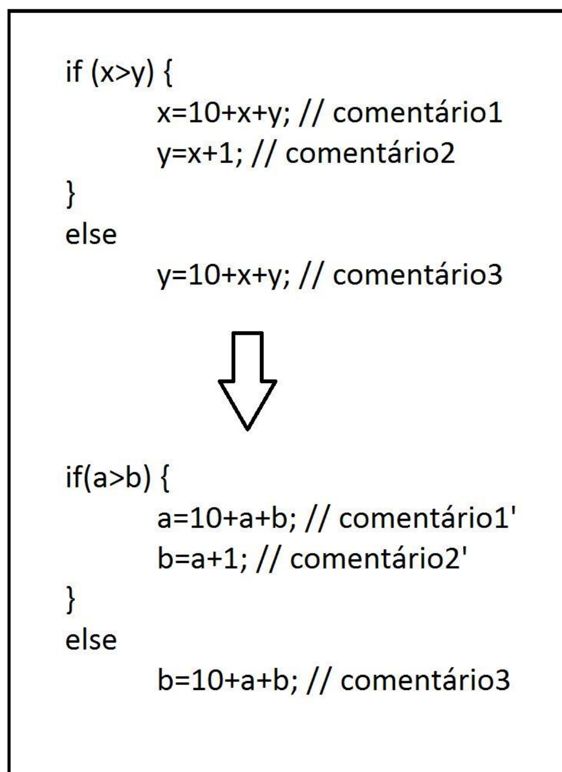


Fonte: Autoria própria

Para o Tipo II, os fragmentos de códigos são idênticos no aspecto semântico. Porém, apresentam modificações na apresentação dos fragmentos. As modificações podem ser nos identificadores, valores literais, tipos, espaçamento e *layout* (ROY; CORDY, 2007). Um exemplo deste tipo de clonagem é ilustrado na

figura 6. Neste esboço é possível perceber que ambos os códigos são idênticos do ponto de vista semântico, mas com algumas alterações na sua forma. As alterações foram mudanças de identificadores e comentários.

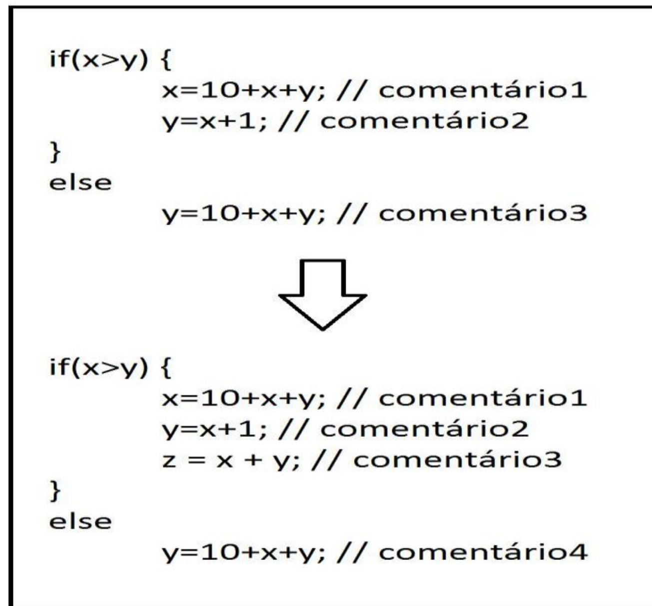
Figura 6 - Plágio de código tipo II



Fonte: Autoria própria

Para o Tipo III de clonagem, as modificações são consideradas mais profundas. Os códigos clonados podem variar por meio de inserção ou remoção de códigos. Os códigos também podem se diferenciar por meio de modificações de identificadores, tipos, valores literais e comentários (ROY; CORDY, 2007). A figura 7 apresenta um exemplo deste tipo de clonagem de código. No exemplo é possível identificar uma inserção de código no código clonado. O código inserido foi o *statement* da atribuição de um valor para a variável *z*. Além da inserção de código, houveram modificações nos comentários.

Figura 7 - Plágio de código tipo III



Fonte: Autoria própria

No tipo IV, os códigos clonados são modificados por meio de variações sintáticas mas executam uma mesma funcionalidade. Este tipo de plágio é o mais difícil de identificar porque as modificações são consideradas mais profundas. Os códigos clonados podem variar por meio de inserção ou remoção de códigos. Em termos sintéticos/estrutural os códigos não são copiados. Os códigos são categorizados por apresentarem similaridade em termos de semântica, em outras palavras, a funcionalidade é a mesma (ROY; CORDY, 2007).

Na figura 8 um exemplo de plágio deste tipo é apresentado e é possível perceber que ambos os códigos apresentados são similares do ponto de vista semântico, ou seja, ambos os códigos apresentam a funcionalidade de calcular o fatorial de um número. Porém, em termos sintáticos as duas implementações são completamente diferentes.

Figura 8 - Plágio de código tipo IV

```
int i, j=1;
  for (i=1; i<=VALUE; i++)
      j=j*i;

            ⇩

int factorial(int n) {
 if (n == 0) return 1 ;
 else        return n * factorial(n-1) ;
}
```

Fonte: Adaptado de (ROY; CORDY, 2007)

De um modo geral, todas os tipos de clonagem de códigos fontes realizam algum mecanismo a fim de esconder o plágio. A dificuldade em identificar estes tipos de plágio se deve ao fato de apresentar geralmente uma combinação de um ou mais tipos de plágio em um código clonado. Uma ferramenta para abranger a verificação de plágio para todos os tipos citados anteriormente tende a se tornar complexa de tal modo a afetar o tempo de processamento da ferramenta.

3.3 TÉCNICAS DE IDENTIFICAÇÃO DE PLÁGIO

O reconhecimento de plágio em códigos-fontes é elaborado a partir de técnicas de identificação de similaridade. Nesta seção, algumas das principais técnicas utilizadas no campo de reconhecimento de plágio em códigos de programação são apresentadas.

3.3.1 Técnica Baseada em Texto

A técnica baseada em texto é uma das mais utilizadas para identificação de plágio em código de programação. Nesta técnica, o código de programação é modelado como um conjunto de linhas ou *strings*. A comparação entre dois conjuntos de linhas ou *strings* é o mecanismo de identificação de similaridade utilizado. Com este nível de similaridade entre dois conjuntos é que se determina o plágio. A comparação funciona de modo a comparar linha por linha ou *string* por *string* (ROY; CORDY, 2007).

Antes de realizar a comparação de dois códigos, em algumas metodologias, é aplicado um filtro sobre o código-fonte a fim de melhorar a precisão da comparação. Segundo Roy e Cordy (2007) os filtros para esta técnica consistem em três diferentes tipos:

- Remoção de comentários: todo e qualquer comentário do código de programação que está sendo filtrado é removido para fazer a comparação.
- Espaço em branco: todo e qualquer espaço em branco entre elementos de um código de programação é eliminado antes de realizar a comparação de similaridade.
- Normalização: um tipo de filtro em que é aplicado por meio de diferentes camadas. Neste caso, o código de programação que está sendo filtrado é passado por diferentes regras de normalização a fim de resultar em um código de programação refinado para a comparação. O intuito é aumentar a precisão de identificação de similaridade.

A técnica baseada em texto utilizando normalização como filtro é exemplificada pela figura 9. Neste esboço, um código de programação é passado por diversas camadas de filtro. Analisando cada uma destas camadas é possível perceber o funcionamento da técnica baseada em texto com normalização. Esta normalização é a do algoritmo *Sherlock*. Segundo MACIEL et al (2012) o funcionamento desta normalização depende de diversas camadas de filtros que são as seguintes:

- Primeira etapa, normalização 1: remoção de linhas e espaços vazios assim como a eliminação de comentários.
- Segunda etapa, normalização 2: remoção de caracteres situados entre aspas.
- Terceira etapa, normalização 3: remoção de valores literais e variáveis.
- Quarta etapa, normalização 4: remoção de todas as palavras reservadas.

Figura 9 - Normalização com o algoritmo *Sherlock*

```
// Imprime resposta
for (i=1;i<= n;i++){

      if (i %3== 1){
            printf("Resp %d", v[i]/x+z);
}
}
```

```
for( i=1; i<=n; i++ ) {
if( i % 3 ==1 ) {
printf ( " Resp %d ", v[ i ] / x +z );
}
}
```

```
for( i=1; i<=n; i++ ) {
if( i % 3 ==1 ) {
printf( , v[ i ] / x +z );
}
}
```

```
for( = ; <= ; ++ ) {
if( % == ) {
printf( , [] / +);
}
}
```

```
( =; <=; ++ ) {
( %== ) {
( , [] / +);
}
}
```

Fonte: Adaptado de MACIEL et al. (2012)

Nesta técnica de identificação de plágio, como observado pela figura 9, é promovido inicialmente uma normalização para que se possa fazer a comparação de dois conjuntos de linhas ou *strings*.

3.3.2 Técnica Baseada em *Token*

A técnica baseada em *token* consiste em transformar um código de programação em uma sequência de conjunto de *tokens*. Um *token*, neste contexto, é a menor unidade de informação tratada por um sistema de detecção de similaridade.

Dependendo do tipo de *token*, uma determinada informação será identificada (LIMA, 2011).

Segundo Roy e Cordy (2007) a grande vantagem da técnica baseada em *token* em relação a baseada em texto é a eficiência em identificar mudanças de códigos. Estas mudanças de códigos podem ser tanto no aspecto de espaçamento como no aspecto de formato de um código modificado.

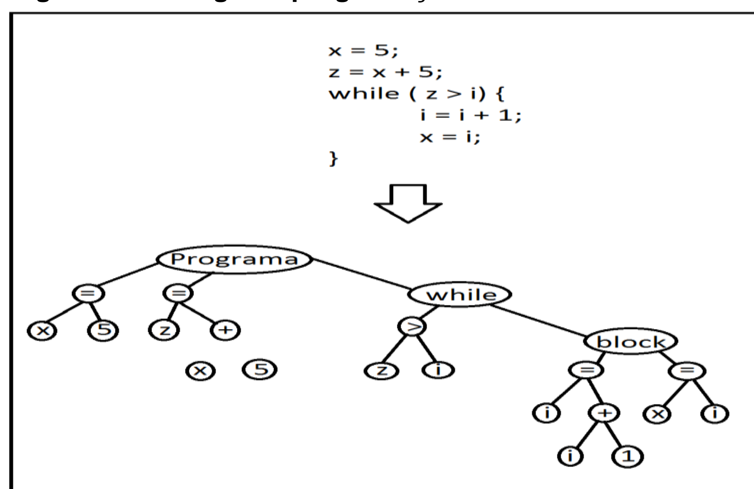
Um exemplo de algoritmo que utiliza uma técnica baseada em *tokens* é o algoritmo de *Baker*. Neste algoritmo, um código de programação é transformado em uma sequência de *tokens* por meio de uma análise léxica. Cada linha do código de programação é mapeada em uma linha contendo apenas *tokens*. Com isso, para cada linha de *token* é feita uma comparação em relação a outros códigos analisados. A complexidade deste algoritmo é $O(n+m)$, onde n é o número de linhas e m é o número de padrões encontradas (ROY; CORDY, 2007).

3.3.3 Técnica Baseada em Árvore

As técnicas baseadas em árvores são modeladas utilizando *parse tree* ou *Abstract Syntex Tree* (AST). Para esta técnica, os códigos fontes são mapeados para estas estruturas. A identificação de plágio é feita por meio da busca de subárvores comuns aos códigos mapeados em uma destas representações. O mapeamento de um código de programação em árvores possibilita uma representação completa do código de programação. Apenas os nomes de variáveis e valores literais não são considerados na representação (ROY; CORDY, 2007).

Segundo Lima (2011), em uma representação AST cada nó e os seus filhos representam um determinado elemento do código de programação. No caso de um nó, a representação é da estrutura de composição da linguagem do código de programação analisado enquanto que os seus filhos são os parâmetros da linguagem. A figura 10 representa uma transformação de um código de programação em uma árvore AST.

Figura 10 - Código de programação em AST



Fonte: Adaptado de (LIMA, 2011)

A figura 10 mostra como um código de programação é convertido em uma estrutura de árvore AST. Após converter os códigos fontes em árvores, inicia a etapa de busca por subárvores similares entre as árvores analisadas.

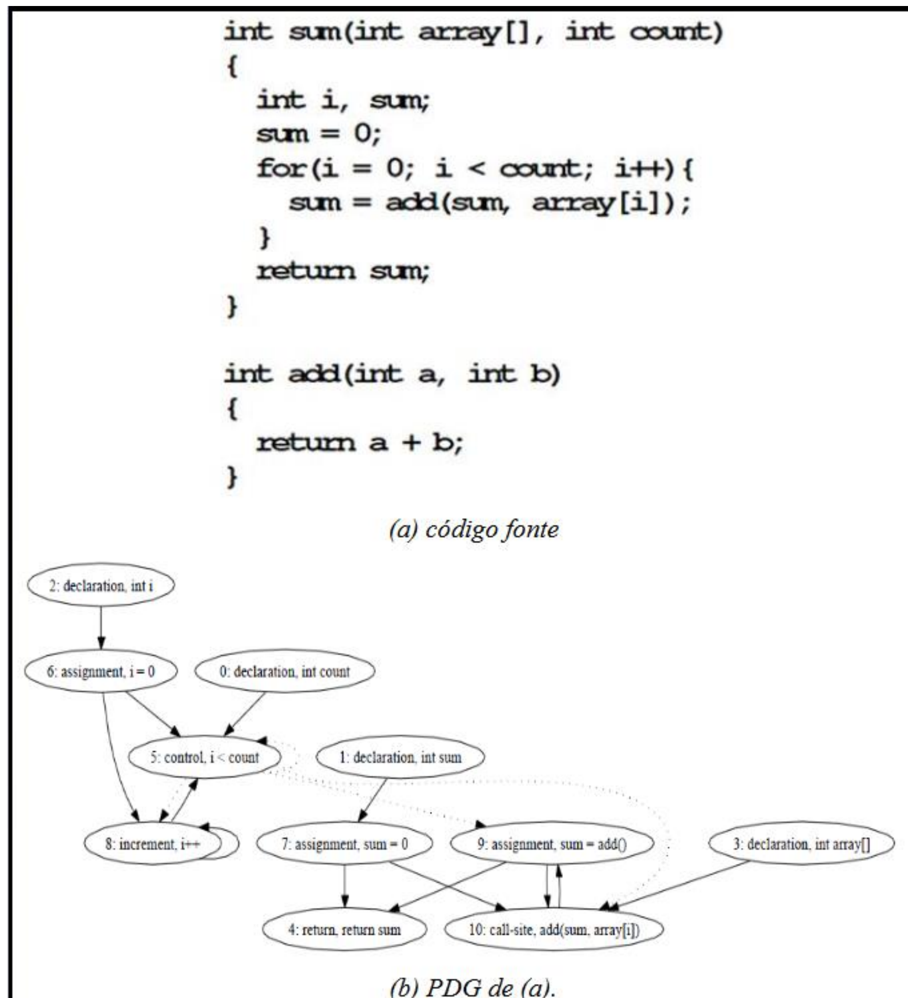
3.3.4 Técnica Baseada em Grafos

É caracterizada por representar um nível de abstração de um código de programação de modo a possibilitar a similaridade semântica entre códigos. Com a representação em grafos, exemplificada na figura 11, é possível mapear os fluxos do código e consequentemente contribui para compreensão semântica dos códigos (ROY; CORDY, 2007).

Segundo Roy e Cordy (2007) a técnica baseada em grafos é eficiente para identificar os seguintes tipos de plágios: reestruturação de instruções, inserção e deleção de código, códigos relacionados e outros tipos de mudanças de códigos de aspecto semântico.

As modificações geralmente realizadas nos códigos plagiados tais como trocar o nome das variáveis ou reordenar as declarações de variáveis são identificadas nesta técnica baseada em grafos. Esta técnica é capaz de tal identificação uma vez que não apresenta informações sintáticas, apenas as relações entre variáveis e operações (LIMA, 2011).

Figura 11 - Código de programação em um Grafo



(a) código fonte

(b) PDG de (a).

Fonte: Lima (2011)

A transformação de um código de programação em um grafo é ilustrada na figura 11. A modelagem em grafos apresenta os nós como sendo as declarações de variáveis e as arestas representam os estados de modificações das mesmas, assim como operações realizadas (LIMA, 2011).

3.3.5 Técnica Baseada em Métricas

Para a técnica de métricas, um código de programação é analisado por unidades sintáticas. Cada unidade sintática pode ser uma classe, método ou até mesmo uma instrução. Uma vez definida a unidade sintática a ser comparada, o próximo passo é definir o conjunto de métricas que será aplicado para a unidade sintática. Como é feita a análise por unidade sintática do código, essa técnica é

aplicada geralmente para um fragmento do código de programação e não para o código inteiro. Um conjunto de métricas pode ser por exemplo: o número de linhas, quantidade de chamadas de funções, quantidade de variáveis dentre outras métricas (ROY; CORDY, 2007).

Com a definição das métricas e das unidades sintáticas prontas é feita uma comparação de métricas para identificar similaridade entre os códigos analisados. As unidades com métricas similares são identificadas como plágio (ROY; CORDY, 2007).

3.4 FERRAMENTAS DE DETECÇÃO DE PLÁGIO

Neste trabalho, duas das principais ferramentas de análise de similaridade de códigos-fontes foram estudadas. A seguir será mostrada uma análise das ferramentas *JPlag* e MOSS (MACIEL, 2014).

A ferramenta *JPlag* foi desenvolvida em *java* e é disponibilizada por meio de *WebService*. O acesso para esta ferramenta é público mas exige um cadastro de autorização para a sua utilização. A implementação desta ferramenta é do tipo código fechado e é capaz de mostrar o nível de similaridade entre códigos-fontes por meio de uma porcentagem (MACIEL, 2014).

Segundo Prechelt, Malpohl, Philippsen (2002), a ferramenta *JPlag* oferece um suporte para verificação de similaridade de códigos-fontes das linguagens de programação: Java, Scheme, C e C++. O funcionamento desta ferramenta é baseado na geração de *tokens* e os códigos-fontes são transformados em *tokens* por meio de um *parser* e um analisador sintático específico para cada tipo de linguagem de programação. Após a geração dos *tokens*, é feita a comparação dos mesmos.

O algoritmo utilizado para a comparação é o "*Greedy String Tiling*". Após as etapas dos *tokens* e das comparações, é apresentado um nível de similaridade. A figura 12 mostra como esta ferramenta utiliza a técnica de *tokens* para códigos-fontes em linguagem java. Neste exemplo, percebe-se a geração de cada *token* para uma determinada estrutura do código de programação que está sendo analisado.

Figura 12 - Funcionamento da ferramenta *JPLAG*

	Java source code	Generated tokens
1	`public class Count {`	BEGINCLASS
2	`  public static void main(String[] args)`	VARDEF,BEGINMETHOD
3	`           throws java.io.IOException {`	
4	`    int count = 0;`	VARDEF,ASSIGN
5		
6	`    while (System.in.read() != -1)`	APPLY,BEGINWHILE
7	`      count++;`	ASSIGN,ENDWHILE
8	`    System.out.println(count+" chars.");`	APPLY
9	`  }`	ENDMETHOD
10	`}`	ENDCLASS

Fonte: Prechelt, Malpohl e Philippsen (2002)

A ferramenta MOSS (*Measure of Software Similarity*) foi desenvolvida em 1994 por Alex Aiken na Universidade da California – Berkeley. Esta ferramenta identifica o nível de similaridade entre códigos-fontes. As linguagens dos códigos-fontes aceitáveis são C, C++, Java, Pascal, ML, LISP, Ada e Scheme (MENAI; AL-HASSOUN, 2010).

O MOSS assim como a ferramenta JPlag, é disponibilizado por meio de *WebService* e necessita de uma autorização para a sua utilização (MACIEL, 2014). O funcionamento do MOSS, exemplificado pela figura 13, é baseado na técnica de *tokens*. Os *tokens* gerados são convertidos em *hashs* para criar uma assinatura (*fingerprint*) de representação do código de programação. Após a conversão em *hashs* dos *tokens* é feita uma seleção de um subconjunto destas estruturas. Para identificar o nível de similaridade, um cálculo da distância entre os *fingerprints* é realizado (SCHLEIMER; WILKERSON; AIKEN apud MACIEL, 2014).

Figura 13 - Funcionamento da ferramenta MOSS

```
A do run run run, a do run run
```
(a) Some text.

```
adorunrunrunadorunrun
```
(b) The text with irrelevant features removed.

```
adoru dorun orunr runru unrun nrunr runru
unrun nruna runad unado nador adoru dorun
orunr runru unrun
```
(c) The sequence of 5-grams derived from the text.

```
77 74 42 17 98 50 17 98 8 88 67 39 77 74 42
17 98
```
(d) A hypothetical sequence of hashes of the 5-grams.

```
(77, 74, 42, 17)   (74, 42, 17, 98)
(42, 17, 98, 50)   (17, 98, 50, 17)
(98, 50, 17, 98)   (50, 17, 98,  8)
(17, 98,  8, 88)   (98,  8, 88, 67)
( 8, 88, 67, 39)   (88, 67, 39, 77)
(67, 39, 77, 74)   (39, 77, 74, 42)
(77, 74, 42, 17)   (74, 42, 17, 98)
```
(e) Windows of hashes of length 4.

```
17 17 8 39 17
```
(f) Fingerprints selected by winnowing.

Fonte: Maciel (2014)

Na Figura 13 a etapa (a) mostra uma entrada de dados qualquer. A etapa (b) realiza a normalização da entrada, ou seja, alguns itens como espaçamento em branco são removidos. A etapa (c) consiste em gerar uma sequência de *substrings* com a entrada após a normalização. A etapa (d) converte cada *substring* em um valor *hash*. A etapa (e) divide os *hashs* em conjuntos de tamanho 4. A etapa (f) mostra um conjunto selecionado como o *fingerprint* da entrada de dados. O critério de seleção é buscar os menores valores *hash* de cada conjunto.

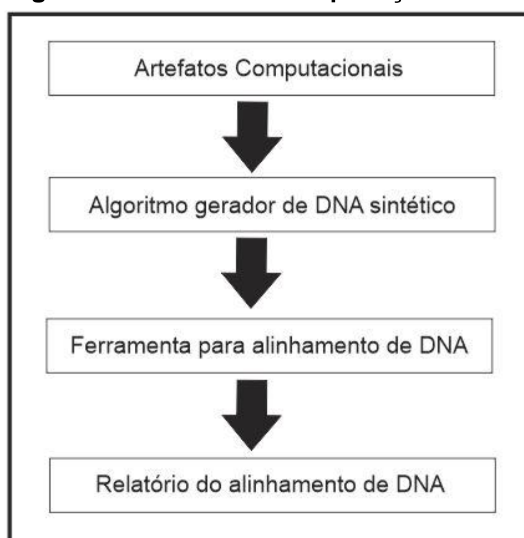
4 APLICAÇÃO DA BIOINFORMÁTICA EM DETECÇÃO DE PLÁGIOS

Este capítulo apresenta o procedimento de aplicação do alinhamento de sequências da bioinformática para identificar plágio em códigos de programação. A abordagem inicial deste capítulo é feita por meio da descrição do processo geral de aplicação que será apresentada na seção 4.1. A seção 4.2 descreve o artefato computacional utilizado no processo. A seção 4.3 mostra o funcionamento de um algoritmo gerador de DNA sintético. A seção 4.4 apresenta a utilização de ferramentas de alinhamento de sequências. Por fim, a seção 4.5 relata uma análise dos relatórios de alinhamentos realizados pelas ferramentas utilizadas.

4.1 PROCESSO GERAL DE APLICAÇÃO DO ALINHAMENTO

Uma metodologia de reconhecimento de *malwares* em computadores por meio de alinhamento de DNA pode ser aplicável em outros domínios da computação tais como identificação de tipos de arquivos e identificação de plágio em códigos de programação (PEDERSEN, 2012). O desenvolvimento deste trabalho apresenta a aplicação desta metodologia no campo de reconhecimento de plágio em códigos de programação. As etapas do processo geral da metodologia proposta por Pedersen (2012) estão ilustradas na figura 14.

Figura 14 - Processo de aplicação da Metodologia



Fonte: Autoria própria

O processo consiste em obter um artefato computacional, que neste trabalho é qualquer código de programação e convertê-lo em um DNA sintético. A conversão do código de programação em uma sequência de DNA é feita por meio de um algoritmo proposto por Pedersen (2012).

Após gerar um DNA sintético, o próximo passo a ser realizado é o alinhamento de sequências. Esse processo recebe como entrada dois ou mais DNA sintéticos. Por último, um relatório é fornecido pela ferramenta de alinhamento de sequências a fim de se obter uma análise de similaridade entre os DNA sintéticos analisados.

Nas próximas seções são detalhadas cada etapa do proposto ilustrado na Figura 14.

4.2 ARTEFATO COMPUTACIONAL

Um artefato computacional nesse trabalho é tratado como códigos de programação. Esses arquivos de programação são usados como entrada de dados para o algoritmo de conversão de um artefato computacional em DNA sintético. Os códigos de programação utilizados nesse trabalho são da linguagem C.

O tipo de linguagem não limita a aplicação da metodologia de Pedersen adotada. Porém, como não existem ferramentas de identificação de plágio para qualquer linguagem, a aplicação da metodologia de Pedersen proposta é para a linguagem C a fim de comparar a sua precisão em identificar plágio de programação em relação a uma ferramenta do mercado. Nesse caso, a comparação foi realizada em relação a ferramenta *JPlag*.

4.3 ALGORITMO GERADOR DE DNA SINTÉTICO

A implementação do algoritmo de Pedersen *et al*. (2012) para converter um artefato computacional em um DNA sintético foi cedida por ele para a realização deste trabalho. O funcionamento do algoritmo é ilustrado pela figura 15. O seu funcionamento é fundamentado em um mapeamento de *bits* em bases nitrogenadas. Os artefatos computacionais (códigos-fontes) de programação são tratados como entradas para esse processo gerador de DNA sintético.

Figura 15 - Mapeamento de bits em bases

Bits	Base
00	T
01	G
10	C
11	A

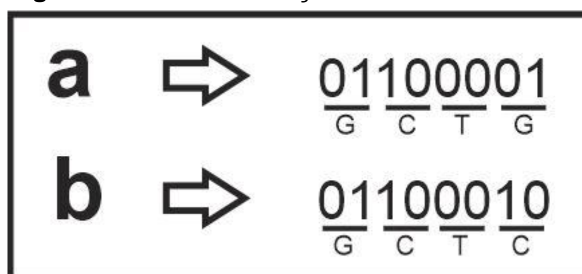
Fonte: Adaptado de PEDERSEN *et al*. (2012)

O funcionamento do algoritmo depende do acesso aos *bytes* que compõem cada entrada. Por exemplo, dado um código de programação em uma determinada linguagem de programação, esse arquivo de programação será a entrada e o próprio algoritmo irá acessar os conjuntos de *bytes* que compõem esse arquivo de entrada. Com o acesso aos *bytes* desse arquivo, é possível fazer um mapeamento, ilustrado pela figura 16, de cada um dos 8 *bits* que fazem parte do *byte* analisado.

O mapeamento, como é mostrado na figura 15, é feito a partir da análise de cada dois *bits* pertencente a um *byte*. Nesse caso, os conjuntos de dois *bits* são: Sétimo e sexto, quinto e quarto, terceiro e segundo e os dois primeiros *bits*. Quando um dos conjuntos possuir valores lógicos de seus *bits* como sendo '0-0' o mapeamento transforma em uma base nitrogenada Timina (T). Para '0-1' será transformado em uma base Guanina (G). No caso de '1-0' a base será Citosina (C) e por último os valores '1-1' transforma na base Adenina (A).

Os *bytes* que compõem cada código de programação são essencialmente um conjunto de *bits* que obedecem a tabela ASCII (*American Standard Code for Information Interchange*). Com isso, cada *caracter* encontrado em um código de programação possui um conjunto de 8 *bits* para a sua identificação no conjunto de *bytes* que compõem o arquivo. Por exemplo, a palavra reservada "main" em linguagem C, é representada por 4 *bytes* de informação. Cada *byte* é referente a uma letra dessa palavra. Considerando como exemplo as letras "*a*" e "b", o mapeamento delas é apresentado na figura 16.

Figura 16 - Transformação em Bases



Fonte: Autoria própria

Como é mostrado na figura 16, se um código de programação tivesse como conteúdo apenas uma letra minúscula ‘a’, esse arquivo possuirá apenas um *byte* e os seus oitos *bits* de acordo com a tabela ASCII são: 01100001. Se o mesmo tivesse apenas como conteúdo a letra ‘b’, o arquivo teria um *byte* do qual seria: 01100010.

Uma observação a ser feita é que todos os arquivos possuem um *byte* destinado ao controle de fim de arquivo, nesse caso conhecido como *End Of Line* (EOF). No contexto desse algoritmo gerador de DNA sintético proposto por Pedersen *et al*. (2012), o controlador de fim de arquivo utilizado é o *Line Feed* (LF). O controlador LF identifica a mudança de linha em um arquivo. Desta forma, sempre o *byte* do controlador de fim de arquivo será acrescentado a todos os arquivos. Portanto, o exemplo do arquivo contendo apenas a letra ‘a’ ou ‘b’ também irá conter um segundo *byte* para indicar o *Line Feed* da única linha contida no arquivo.

4.4 FERRAMENTA PARA ALINHAMENTO DE DNA

Nessa etapa do processo de identificação de plágio por meio de alinhamento de sequências são utilizadas ferramentas próprias de alinhamento de sequências. As ferramentas utilizadas nesse trabalho são EMBOSS e BLAST. Independente da ferramenta utilizada, a entrada exigida por uma ferramenta de alinhamento são duas ou mais sequências. Nesse contexto de aplicação, as entradas são tratadas como sendo os DNA sintéticos criados a partir do algoritmo de gerador de DNA sintético, descrito na Seção 4.3.

Com o DNA sintético de cada código de programação a ser analisado, um alinhamento é realizado pela ferramenta escolhida. No caso o alinhamento pode ser local ou global. Nesse trabalho está sendo avaliado o desempenho de um alinhamento do tipo global para a identificação de similaridade entre duas ou mais

sequências. O alinhamento do tipo local busca por diversas regiões similares presentes entre as sequências comparadas. A figura 17, ilustra de uma maneira geral o funcionamento do alinhamento de DNA a partir de uma ferramenta, nesse caso a EMBOSS.

Figura 17 – EMBOSS Alinhamento de DNA sintético

```
EMBOSS_001         1 GCTGTTCC
                       |||.||||
EMBOSS_001         1 GCTCTTCC
```

Fonte: Autoria própria

O exemplo da figura 17 exibe que foram utilizados dois códigos de programação com diferentes conteúdos para mostrar como funciona o alinhamento de DNA sintético. A sequência GCTGTTCC é referente a um código que possui como conteúdo a letra ‘a’ e a segunda sequência GCTCTTCC é referente a um código de programação que possui como conteúdo a letra ‘b’. Observe que as bases ‘TTCC’ no final de cada sequência representa o *Line Feed*. O alinhamento é feito obedecendo as regras de pontuações baseadas em casos de *matches*, *mismatches* e *gaps*. O alinhamento realizado foi do tipo global.

4.5 RELATÓRIO DO ALINHAMENTO DE DNA

O processo de alinhamento gera como resultado um relatório de informações pertinentes ao alinhamento realizado. As principais informações fornecidas por esses relatórios são as seguintes: porcentagem de similaridade, porcentagem de identidade, *score* e *e-value*. Todas essas informações são fornecidas pelas ferramentas de alinhamento estudadas nesse trabalho com exceção do *e-value* fornecido apenas pela BLAST. As ferramentas estudadas foram BLAST e EMBOSS. A figura 18 ilustra um relatório de alinhamento gerado pela BLAST, onde mostra todas as informações importantes que foram analisadas nesse trabalho.

Figura 18 – Relatório BLAST de Alinhamento de DNA sintético

```
Database: tipo1A
           1 sequences; 768 total letters



Query= lcl|/home/manager/tipo1B.c

Length=804
                                                                      Score     E
Sequences producing significant alignments:                          (Bits)  Value

lcl|/home/manager/tipo1A.c                                             599    2e-175


>lcl|/home/manager/tipo1A.c
Length=768

 Score =  599 bits (324),  Expect = 2e-175
 Identities = 324/324 (100%), Gaps = 0/324 (0%)
 Strand=Plus/Plus

Query  1    TCTAGCCGGCACGCTAGCATGAGGGCGTGCGGTCTTTAATGATAGAGTGCGTGCCGGCAA  60
            ||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||
Sbjct  1    TCTAGCCGGCACGCTAGCATGAGGGCGTGCGGTCTTTAATGATAGAGTGCGTGCCGGCAA  60
```

Fonte: Autoria própria

As porcentagens de similaridade e identidade são comumente utilizadas como sendo medidas iguais pela maioria das ferramentas de alinhamento. Porém, as suas definições são distintas. Considerando o contexto de sequências de DNA sintético, a porcentagem de identidade se refere a quantidade de bases nitrogenadas (C, G, A, T) idênticas entre as sequências comparadas. A sua fórmula pode ser calculada por meio da relação entre o número de bases iguais e o tamanho da menor sequência entre as analisadas.

A definição de porcentagem de similaridade está relacionada com o grau de facilidade em transformar uma sequência na outra. Quanto maior o nível de similaridade, menos esforço é necessário para realizar a transformação. O processo de mudar as sequências é feito a partir de um conjunto de operações de manipulação de bases nitrogenadas para sequências do tipo DNA. As operações são as seguintes: inserção, remoção e substituição. As ferramentas de alinhamento utilizadas nesse trabalho não diferenciam essa medida em relação a taxa de identidade encontrada, ambas se referem a um mesmo valor.

O valor de *score* depende do sistema de pontuação adotado. O sistema de pontuação consiste em atribuir uma determinada pontuação para cada uma das seguintes situações: *matches*, *mismatches* e *gap*. O valor *score* é a soma das

pontuações atribuídas. Esse valor é comumente utilizado para comparar os resultados de alinhamentos.

O *e-value* é um parâmetro que envolve probabilidade e mede a chance de o alinhamento realizado encontrar sequências iguais em uma determinada base de dados. No contexto de identificação de plágio, pode ser entendido como verificar plágio de um código de programação em relação a um banco de dados com diversos outros códigos.

5 RESULTADOS

Este capítulo apresenta os resultados gerados a partir da aplicação da bioinformática no reconhecimento de plágio em códigos de programação. Um conjunto de testes foi realizado para avaliar essa nova proposta de reconhecimento de plágio. Para cada teste foi realizada uma comparação dos resultados gerados pela nova proposta em relação aos resultados da ferramenta JPLAG. A seção 5.1 apresenta um conjunto de testes para cada tipo de plágio relatado nesse trabalho. A seção 5.2 mostra a análise dos conjuntos de testes, identificando pontos importantes que justificam os resultados encontrados. A seção 5.3 sugere uma proposta de mudanças na metodologia de bioinformática adotada para identificação de plágio em códigos de programação.

5.1 CONJUNTO DE TESTES

Para identificar a possibilidade de aplicação desta nova técnica de identificação de plágio baseada na bioinformática, um conjunto de teste foi elaborado para avaliar a precisão na tarefa de identificar plágios de diferentes tipos. O conjunto de testes foi composto de 18 códigos em linguagem C. Um total de 8 códigos, utilizados para identificar cada tipo de plágio, foram retirados do capítulo 2. Outros 10 códigos foram testados adicionalmente, do qual 6 pertencem ao apêndice A. Um resumo do resultado encontrado para cada teste é ilustrado pela figura 19.

Figura 19 – Resultados obtidos

Conjunto de teste	Resultado
Plágio **tipo I**	Metodologia = **72.6%**
	JPLAG = **90%-100%**
Plágio **tipo II**	Metodologia = **94.9%**
	JPLAG = **90%-100%**
Plágio **tipo III**	Metodologia = **80.5%**
	JPLAG = **0%-10%**
Plágio **tipo IV**	Metodologia = **49.8%**
	JPLAG = **0%-10%**

Fonte: Autoria própria

Para a identificação de plágio do tipo I, foram utilizados os códigos de programação em linguagem C ilustrados pela figura 5. Ao tentar identificar plágio do tipo I com esse exemplo de código utilizado, busca-se avaliar a capacidade da metodologia de Pedersen *et al.* (2012) em reconhecer técnicas de espaçamento, variações de comentários e a mistura de ambas anteriormente citadas.

O resultado encontrado na identificação de plágio do tipo I foi baseado na execução de duas ferramentas JPLAG e a metodologia de Pedersen *et al.* (2012) baseada em bioinformática. A ferramenta JPLAG apresentou como resultado uma faixa de similaridade entre 90% e 100% para esse conjunto de teste. Enquanto a metodologia de Pedersen *et al.* (2012) apresentou como resultado uma taxa de 72.6% de similaridade.

No caso da identificação de plágio do tipo II, foram utilizados os códigos de programação em linguagem C ilustrados pela figura 6. O principal objetivo nesse teste realizado foi avaliar a capacidade de identificação de códigos similares no aspecto semântico, porém com uma apresentação visual do código diferente por meio de mudança nos identificadores. Nesse exemplo de código utilizado, os códigos tiveram as suas variáveis renomeadas e algumas mudanças no conteúdo dos comentários também foram efetuadas.

Ao avaliar o resultado da etapa de identificação de plágio do tipo II, duas ferramentas foram utilizadas: JPLAG e a metodologia de Pedersen *et al.* (2012) baseada em bioinformática. A ferramenta JPLAG identificou uma taxa entre 90% e 100% de similaridade. A metodologia de Pedersen *et al.* (2012) considerou uma taxa de similaridade de 94.9%.

O conjunto de teste, realizado para identificar o plágio do tipo III, utilizou os códigos de programação em linguagem C ilustrados pela figura 7. O intuito principal nessa avaliação de plágio do tipo III foi identificar códigos de programação plagiados por meio de inserção de trecho de códigos.

Após realizar teste para a identificação de plágio do tipo III através da amostra de códigos da figura 7, anteriormente citada, obteve-se resultados de similaridade com a utilização das seguintes ferramentas: JPLAG e a metodologia de Pedersen *et al.* (2012) baseada em bioinformática. A ferramenta JPLAG conseguiu constatar um índice de similaridade na faixa de 0% até 10%. Por outro lado, a

metodologia baseada em bioinformática identificou uma taxa de similaridade de 80.5%.

Para o último conjunto de teste, o de identificação do plágio do tipo IV, os códigos de programação da figura 8 foram utilizados como amostras para o uso das ferramentas. Nesse teste, o nível de complexidade é maior devido a possibilidade de existir plágio entre dois códigos de programação com aparências completamente distintas. Para esse tipo de plágio, pode ser utilizada qualquer técnica de camuflagem de plágio. O principal objetivo nesse teste foi identificar plágio em termos semântico e não sintático.

O último resultado encontrado, nesse caso relacionado ao teste de identificação de plágio do tipo IV, as ferramentas JPLAG e a metodologia de Pedersen *et al.* (2012) baseada em bioinformática foram utilizadas. A ferramenta JPLAG conseguiu identificar uma taxa de similaridade entre 0% e 10%. Por outro lado, a metodologia de Pedersen *et al.* (2012) baseada em bioinformática apresentou um índice de similaridade de 49.8% para as amostras avaliadas.

Outros conjuntos de testes foram aplicados neste trabalho, os mesmos se encontram no Apêndice A. Nesse conjunto de teste adicional, foi realizado três testes diferentes. Os conjuntos de testes foram nomeados como: A, B e C. O conjunto de teste A mostra mais um caso de plágio do tipo II. Os conjuntos de testes B e C foram considerados como exemplos de códigos não plagiados.

5.2 ANÁLISE DOS CONJUNTOS DE TESTES

Os conjuntos de testes aplicados nesse trabalho foram essenciais para identificar questões pontuais quanto a precisão da metodologia de Pedersen *et al.* (2012) em reconhecer diversos tipos de plágios de programação. Uma análise a seguir será feita para os diferentes tipos de testes aplicados, sendo eles: tipo I, II, III e IV.

O teste, aplicado para a identificação de plágio do tipo I, mostra a capacidade da metodologia de Pedersen *et al.* (2012) em reconhecer plágio dessa categoria. O índice de similaridade encontrado foi de 72.6%. No entanto, essa taxa foi abaixo da encontrada pela ferramenta JPLAG, que reconheceu o código como tendo uma taxa entre 90% e 100%.

A diferença encontrada nos resultados de identificação de plágio do tipo I mostra alguns problemas da metodologia de Pedersen *et al*. (2012). A taxa de apenas 72.6% encontrada pela metodologia de Pedersen *et al*. (2012) indica a sua dificuldade em identificar plágio diante de técnicas de camuflagem simples como espaçamento, quebra de linhas, mudanças de comentários e outras formas de alterações no visual do fragmento de código.

Essa incapacidade indicada anteriormente, está relacionada com a maneira como um código é transformado em DNA sintético. Existe um mapeamento de todo e qualquer caractere *ASCII* em bases nitrogenadas. Com isso, a quantidade total de linhas, espaçamento e comentários podem influenciar na criação do DNA sintético. Sendo assim, nesse exemplo de código plagiado tipo I, a reduzida taxa de similaridade de 72.6% em relação a outras ferramentas se justifica em função da existência dessas técnicas de camuflagem e a forma como um DNA sintético é gerado.

O resultado, do teste aplicado para a identificação de plágio tipo II, demonstra a eficiência da metodologia de Pedersen *et al*. (2012) em reconhecer importantes técnicas de plágio. As técnicas analisadas nesse contexto foram: mudança de identificadores assim como os seus valores literais.

No teste de plágio do tipo II, as mudanças nas variáveis e nos seus respectivos valores literais foram incapazes de burlar a metodologia de Pedersen *et al*. (2012). Com um resultado similar da ferramenta JPLAG, a metodologia identificou 94.9% de similaridade nos códigos de programação analisados. Tal resultado mostra a eficiência em analisar situações de códigos plagiados por meio dessas técnicas citadas.

Em relação ao teste realizado para a identificação de plágio do tipo III, alterações em códigos por meio da técnica de inserção de trechos de códigos foram analisadas. A ferramenta JPLAG não conseguiu identificar o plágio contido nos códigos de programação comparados. Sendo assim, o JPLAG considerou uma taxa de plágio entre 0% e 10%, a taxa mais baixa que a ferramenta pode avaliar. Por outro lado, a metodologia de Pedersen *et al*. (2012) reconheceu uma taxa de similaridade de 80.5% para esse mesmo caso.

De acordo com o teste de plágio do tipo III, a metodologia de Pedersen *et al*. (2012) apresentou vantagem em relação a ferramenta JPLAG. A mesma foi capaz de identificar inserção proposital em código a fim de camuflar códigos plagiados. A

sua eficiência nesse quesito é justificada pelo fato de levar em consideração todos os caracteres ASCII presentes nos códigos. Diferente de técnicas como a do *token* que pode ignorar algum trecho de código, a metodologia de Pedersen *et al.* (2012) não é suscetível a esse comportamento falho, como nesse caso.

Para o teste de plágio do tipo IV, a tarefa de identificar plágio é mais complexa, onde todas as técnicas de plágios podem ser utilizadas. Nesse conjunto de teste foi avaliada a capacidade de identificar plágio semântico nos códigos de programação. A ferramenta JPLAG não considerou os códigos analisados como sendo plagiados, desta forma, considerou uma taxa de similaridade entre 0% e 10%. Contudo, a metodologia de Pedersen *et al.* (2012) encontrou uma taxa de similaridade de 49.8% presente nos exemplos analisados.

O resultado encontrado pela metodologia de Pedersen *et al.* (2012) na identificação de plágio semântico não pode ser considerado eficiente. Ainda considerando uma taxa de 49.8%, ao analisar como a metodologia de Pedersen *et al.* (2012) encontrou essa taxa, pode-se perceber um nível aleatoriedade presente nesse resultado. O nível de aleatoriedade em questão está relacionado com o fato de a técnica de geração do DNA sintético ser fundamentada no mapeamento de apenas 4 bases nitrogenadas.

O fato da metodologia de Pedersen *et al.* (2012) ser baseada em 4 bases nitrogenadas, limita a diferenciação das informações identificadas durante o processamento do binário do código de programação analisado. Como o arquivo é mapeado a cada conjunto de dois *bits*, qualquer que seja a informação contida no código de programação, a mesma será convertida necessariamente em uma das bases nitrogenadas. Desta forma, mesmo que os códigos sejam diferentes em conteúdo, a metodologia de Pedersen *et al.* (2012) acaba identificando uma porcentagem mínima de similaridade. Os conjuntos de testes B e C contidos no Apêndice A, mostram que códigos não plagiados, considerados completamente diferentes, apresentam sempre essa taxa mínima de similaridade.

5.3 PROPOSTAS DE MUDANÇAS NA METODOLOGIA BASEADA EM BIOINFORMÁTICA

A nova proposta de identificação de plágio baseada em bioinformática apresentou resultados aceitáveis no processo de aplicação dos conjuntos de testes. Ainda considerando a sua eficiência na detecção de tipos específicos de plágios, para determinadas situações a metodologia de Pedersen *et al*. (2012) não apresenta resultados adequados. Algumas melhorias podem ser incorporadas à nova metodologia, de modo a corrigir suas possíveis falhas e tornar a identificação de similaridade mais precisa.

A figura 20 apresenta um pseudocódigo das propostas de mudanças na metodologia de Pedersen *et al*. (2012).

Figura 20 - Pseudocódigo da proposta de mudanças

```
relatorio funcaoIdentificarPlagio ( arquivo codigo-fonte1, arquivo código-fonte2)

    arquivo novo-codigo-fonte1
    arquivo novo-codigo-fonte2

    novo-codigo-fonte1 = filtrarArquivo( codigo-fonte1)
    novo-codigo-fonte2 = filtrarArquivo (codigo-fonte2)

    FASTA seq1 = geradorDNASintetico( novo-codigo-fonte1)
    FASTA seq2= geradorDNASintetico( novo-codigo-fonte2)

    relatorio resultado = BLAST( seq1, seq2)

    retorna resultado
```

Fonte: Autoria própria

A possiblidade de aplicar um filtro de normalização na etapa de geração de DNA sintético pode contribuir para uma maior precisão em plágios do tipo I. Aspectos relacionados a espaçamentos, quebras de linhas e comentários poderiam ser filtrados por meio da função “filtrarArquivo()” da figura 20, de modo a não serem contabilizados no mapeamento de seus respectivos *bits* no arquivo binário do código de programação em questão.

Uma outra modificação com o intuito de aprimoramento na metodologia de Pedersen *et al*. (2012), seria trocar o tipo de alinhamento de DNA adotada. O presente trabalho adotou alinhamento do tipo global para compor a nova proposta de identificação de plágio. Contudo, ao utilizar um alinhamento do tipo local com modificações na etapa final de geração de relatório, resultados potencialmente mais precisos podem ser encontrados. Esse processo de utilizar um alinhamento local modificado é compreendido pela função “BLAST()” da figura 20.

A justificativa para adotar um diferente tipo de alinhamento está associada com a forma que o alinhamento local se diferencia do global. O alinhamento global é adotado na metodologia de Pedersen *et al*. (2012) e com isso todo o código é levado em consideração na taxa de similaridade. O alinhamento local, poderia analisar a *priori* apenas as diversas partes semelhantes no código. Além disso, uma reformulação nos cálculos de similaridade precisa ser desenvolvida para acompanhar essa mudança na estrutura do tipo de alinhamento adotado.

Ao utilizar o alinhamento local, diversas partes similares entre diferentes códigos de programação podem ser identificadas. Para determinar a taxa de similaridade seguindo esse modelo, seria necessário representar o quanto dessas partes similares representam a totalidade do código de programação. Dessa maneira, poderia chegar por meio de cálculos específicos, a uma taxa de similaridade mais precisa em relação a forma atual como a metodologia de Pedersen *et al*. (2012) está estruturada.

Um outro aspecto falho da metodologia de Pedersen *et al*. (2012), o problema do nível de aleatoriedade na identificação de plágio do tipo IV, pode ser reduzido com uma maior quantidade de informações representadas. A forma atual da metodologia se fundamenta na combinação de dois *bits* para mapear em uma das 4 bases nitrogenadas.

O fato de possuir apenas 4 tipos de informações mapeadas, as bases nitrogenadas, limita no momento de analisar dois códigos completamente diferentes sintaticamente. Essa limitação faz a metodologia de Pedersen *et al*. (2012) sempre apresentar uma taxa de similaridade mínima, a partir de um determinado tamanho de código de programação. Ao aumentar o número de mapeamentos de informações essa limitação seria reduzida, porém, não eliminada completamente. O tamanho do código continua sendo um limitante para esse tipo de problema demonstrado. Desta forma, um novo sistema de mapeamento dos bits em diferente de apenas considerar 4 bases nitrogenadas poderia ser aplicado. A função “geradorDNASintetico()” da figura 20 ilustra essa questão.

6 CONCLUSÃO

Neste trabalho foi apresentado um novo conceito de identificação de plágio baseado em bioinformática. Uma forma diferente de identificar plágio em códigos de programação, pois analisa o arquivo binário dos mesmos. Desta forma, a possiblidade de eficiência em reconhecimento de diferentes tipos de plágios, assim como, a sua flexibilidade pode tornar esse novo conceito uma tendência para as ferramentas atuais de identificação de plágio.

A metodologia de Pedersen *et al*. (2012) baseada em bioinformática apresenta algumas etapas para a sua aplicação. As principais etapas envolvidas são as seguintes: Geração de DNA sintético, alinhamento de DNA e análise de relatório de alinhamento.

A etapa inicial é relacionada com a ideia de abstrair um artefato computacional em um DNA sintético, nesse caso, o artefato computacional foi um código de programação. O desenvolvimento de um algoritmo para realizar a criação de um DNA sintético a partir de um código de programação é essencial. O algoritmo gerador de DNA sintético foi responsável por realizar um mapeamento dos *bits* do arquivo binário de um código de programação em bases nitrogenadas. O fato de utilizar bases nitrogenadas dá-se a possiblidade de utilizar ferramentas de bioinformática de alinhamento. A possibilidade de trabalhar com ferramentas de bioinformática implica em facilidade de realização de um alinhamento. A dificuldade nessa etapa se concentra no algoritmo gerador de DNA sintético.

A segunda etapa é a realização de um alinhamento de DNA a partir dos DNA genéticos criados. Essa etapa é o momento em que duas sequências são comparadas a partir de um modelo comparador baseado em conceitos da bioinformática. Essa comparação busca por similaridades entre as duas sequências. Após realizar um alinhamento, um relatório dessa comparação é gerado pela ferramenta de bioinformática adotada, nessa aplicação foi utilizada a EMBOSS. A facilidade, nesta etapa, está relacionada com a utilização de diversos recursos das principais ferramentas de bioinformática para alinhamento. Podendo ser possível inclusive ter acesso ao código-fonte de algumas delas. No entanto, a critério de dificuldade nesta etapa pode ser considerara a complexidade dos algoritmos de

alinhamento das ferramentas. Desta forma, o tamanho da entrada é um fator limitante.

A última etapa principal é a análise do relatório de alinhamento de DNA realizado pela ferramenta de bioinformática. Diversas informações são utilizadas para determinar a proximidade entre dois DNA sintáticos. Com o intuito de determinar a taxa de similaridade de códigos de programação, informações relacionadas a taxa de identidade foram consideradas nesse trabalho. A facilidade nessa etapa está em ter acesso a um relatório completo do alinhamento. Diversas informações do relatório são fundamentais para identificar o nível de similaridade das sequências alinhadas. Porém, a dificuldade nessa etapa consiste em identificar qual parâmetro do relatório pode ser julgado no processo de identificação de similaridade, no aspecto de plágio em códigos de computadores.

Um conjunto de testes foi aplicado para avaliar a precisão da metodologia de Pedersen *et al.* (2012) em identificar diferentes tipos de plágios. Também, foi considerada uma ferramenta de identificação de plágio, a JPLAG, para comparar os resultados de uma técnica já consolidada no ramo de plágio em relação a metodologia de Pedersen *et al.* (2012) baseada em bioinformática.

Os resultados encontrados nessa fase de teste indicam a eficiência da nova proposta em identificar diferentes tipos de plágio. Os testes para a verificação de plágio dos tipos I e II apresentaram resultados semelhantes em relação a metodologia e a ferramenta JPLAG. Por outro lado, os testes de identificação de plágio dos tipos III e IV apresentaram resultados completamente distintos, favorecendo a metodologia de Pedersen *et al.* (2012) em relação a ferramenta JPLAG. Alguns pontos críticos como: sensibilidade na identificação de variação de espaçamentos, comentários e presença de taxa mínima de similaridade em códigos distintos foram levantados para melhor compreender e aprimorar esses resultados apresentados pela metodologia de Pedersen *et al.* (2012).

6.1 TRABALHOS FUTUROS

Para continuar os estudos relacionados a aplicação de bioinformática na identificação de plágio, sugere-se aplicar as sugestões de melhorias levantadas

neste trabalho para a metodologia de Pedersen *et al*. (2012). Desta forma, um novo trabalho seria sugerido.

Um conjunto de teste para continuar validando a metodologia de Pedersen *et al*. (2012) ou uma nova metodologia baseada no aprimoramento desta é de fundamental importância. A ideia é testar as ferramentas e a metodologia em cenários de códigos de programação considerados extensos, aqueles com centenas ou milhares de linhas de código.

Como um trabalho auxiliar, após a consolidação da metodologia de Pedersen *et al*. (2012) ou até mesmo uma nova metodologia baseada no aprimoramento desta, seria importante a criação de uma plataforma *web* de identificação de plágio em códigos de computadores. A plataforma seria baseada nesse modelo de identificar plágio utilizando a bioinformática.

REFERÊNCIAS

BLAST. **Basic Local Alignment Search Tool**. Disponível em: <http://www.ncbi.nlm.nih.gov/books/NBK279690>. Acesso em: Junho 2016.

BRITO, R, T. **Alinhamento de sequências Biológicas.** 2003. 181 f. Programa de Mestrado em Ciência da Computação - Universidade de São Paulo. São Paulo, 2003.

DICTIONARY.CAMBRIDGE.ORG. **Cambridge Dictionary**. Disponível em: <http://dictionary.cambridge.org/pt/dicionario/ingles/plagiarize>. Acesso em: Novembro 2016.

HUNTER, E. L. **The processes of life:** an introduction to molecular biology. Massachusetts: Massachusetts Institute of Technology Press, 2009. 499p.

KIKUCHI, H.; et al. A Source Code Plagiarism Detecting Method Using Sequence Alignment with Abstract Syntax Tree Elements. In: THE 15TH IEE/ACIS INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, ARTIFICIAL INTELLIGENCE, NETWORKING AND PARALLEL/DISTRIBUTED COMPUTING, 15., 2014, Las Vegas. **Proceedings…** Las Vegas: SNPD’14, 2014. 16p.

LIMA, E, D, C. **Análise de Técnicas e Ferramentas de Detecção de Plágio, e Desenvolvimento de um Protótipo de nova Ferramenta**. 2011. 88f. Trabalho de Conclusão de Curso (Sistemas de Informação) – Universidade Federal de Lavras, Lavras, 2011.

MACIEL, D, L.; et al. Análise de similaridade de códigos-fonte como estratégia para o acompanhamento de atividades de laboratório de programação. In: Centro Interdisciplinar de Novas Tecnologias na Educação, 12., 2012, Porto Alegre. **Anais…** Porto Alegre: CINTED-UFRGS, 2012. p. 2-4.

MACIEL, D, I. **Sherlock N-Overlap: Normalização invasiva e coeficiente de sobreposição para análise de similaridade entre códigos-fonte em disciplinas de programação**. 2014. 105 f. Dissertação (Mestrado em Engenharia de Teleinformática) - Departamento de engenharia de Teleinformática, Universidade Federal do Ceará, Fortaleza. 2014.

MENAI, M, E, B.; AL-HASSOUN, N, S. Similarity Detection in Java Programming Assignments. In: THE 5TH INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE & EDUCATION, 5., 2010, China. **Proceedings…** China: 2012. 7p.

MERRIAM-WEBSTER. **Merriam webster Dictionary**. Disponível em: <http://www.merriam-webster.com/dictionary/plagiarism>. Acesso em: Novembro 2016.

NCBI. **National Center for Biotechnology Information**. Disponível em: <http://www.ncbi.nlm.nih.gov>. Acesso em: Junho 2016

NETO, J. P. Comparação de sequências de DNA. **Revista Científica da Faculdade Lourenço Filho**, Fortaleza, v. 7, n. 1, 2010.

PLAG-PT. Disponível em: <https://www.plag.pt/estatisticas-de-plagio>. Acesso em: jun. 2016.

PRECHELT, L.; MALPOHL, G.; PHLIPPSEN, M. JPlag: Finding Plagiarisms among a Set of Programs. 2000. **Facultat fur Informatik, Universitat Karlsruhe**, n.1, 2000.

PEDERSEN, J.; *et al*. Blast your way through malware analysis assisted by bioinformatics tools. In: THE 2012 WORLD CONGRESS IN COMPUTER SCIENCE, COMPUTER ENGINEERING AND APPLIED COMPUTING, 12., 2012, Las Vegas. **Proceedings…** Las Vegas: WORLDCOMP’12, 2012. 7p.

ROY, C. K; CORDY, J. R. A Survey on Software Clone Detection Research. School of Computing Queen’s University, Kingston. Set. 2007. 109p. **Relatório Técnico**.

SANTOS, E, C. **Uma introdução a Bioinformática através da análise de algumas ferramentas de *software* livre ou código aberto utilizando para o estudo de alinhamento de sequências**. 2004. 73f. Dissertação (Especialista em Administração em Redes Linux) - Departamento de Ciência da Computação, Universidade Federal de Lavras, Lavras. 2004.

SCHLEIMER, S.; WILJERSON, D. S.; AIKEN, A. Winnowing: local algorithms for document fingerprinting. In The 2003 ACM SIGMOD International Conference on Management of Data, 03., 2003, New York. **Proceedings…** New York: SIGMOD ’03, 2003. 9p.

VIANA, G, V. R.; MOURA, H, A, S. Algoritmos para Alinhamento de Sequências. **Revista Multidisciplinar da Uniesp**, São Paulo, n. 6, dez. 2008.

VERLI, H. Bioinformática: Da Biologia à Flexibilidade Molecular. **Sociedade Brasileira de Bioquímica e Biologia Molecular**, São Paulo, ed. 1, dez. 2014. 282p.

WATERMAN, M. S. **Introduction to Computational Biology**: maps, sequences and genomes. 1 ed. The United States: Chapman and Hall, 1995. 448 p.

APÊNDICE A – Conjunto de testes adicionais

Este apêndice apresenta os conjuntos de testes e seus respectivos resultados.

Figura 21 – Conjunto de teste A - Código 1

```
public class BuscaLargura {

    public void realizarBusca(Vertice raiz, Vertice destino) {

        System.out.println("");
        System.out.println("Busca em profundidade");
        System.out.println("Caminho percorrido = ");

        raiz.setPai(raiz);
        int calc = 0;
        ArrayList<Vertice> vertices = new ArrayList<Vertice>(); // controlar o caminho percorrido
        ArrayList<Vertice> caminho = new ArrayList<Vertice>(); // caminho definido, se encontrar.
        Queue<Vertice> fila = new LinkedList<>();
        raiz.setVisitado(true);
        fila.add(raiz);

        while( !fila.isEmpty()) {
            Vertice verticeAtual = fila.remove();
            vertices.add(verticeAtual);
            System.out.print(verticeAtual.getNomeVertice()+" ");
            if ( verticeAtual.getNomeVertice().equals(destino.getNomeVertice())) {

                String nome = verticeAtual.getPai().getNomeVertice();
                calc = verticeAtual.getCusto() + calc;
                caminho.add(verticeAtual);
                while( !nome.equals(raiz.getNomeVertice())) {

                    for ( int i = 0; i < vertices.size(); i++) {

                        if ( vertices.get(i).getNomeVertice().equals(nome)) {
                            //System.out.println("entrou no if do for do wwhile");
                            //System.out.println(vertices.get(i).getNomeVertice());

                                nome = vertices.get(i).getPai().getNomeVertice(); // atualiza quem Ã© o pai
                                caminho.add(vertices.get(i)); // guarda o vertice no caminho
                                calc = vertices.get(i).getCusto() + calc;


                        }
                    }
                }
                caminho.add(raiz);
                break;
            }

            for( int i = 0; i < verticeAtual.getVizinhos().size(); i++) {
                if ( !verticeAtual.getVizinhos().get(i).isVisitado()) {
                    //System.out.println(verticeAtual.getVizinhos().get(i).getNomeVertice());
                    verticeAtual.getVizinhos().get(i).setVisitado(true);
                    verticeAtual.getVizinhos().get(i).setPai(verticeAtual); // setar nome do pai
                    verticeAtual.getVizinhos().get(i).setCusto(verticeAtual.getPesosVizinhos().get(i)); // setar custo
                    fila.add(verticeAtual.getVizinhos().get(i)); // fila de controle
                }
            }
        }
        System.out.println("");
        System.out.println("Caminho solucao = ");
        for ( int indice = caminho.size()-1; indice >= 0; indice--) {
            System.out.print(caminho.get(indice).getNomeVertice()+" ");
        }
        System.out.println("Custo = "+calc);
    }
}
```

Fonte: Autoria própria

Figura 22 - Conjunto de teste A - Código 2

```
public class BuscaProfundidade {

    public void realizarBusca(Vertice raiz, Vertice destino) {

        System.out.println("");
        System.out.println("Busca em profundidade");
        System.out.println("Caminho percorrido = ");

        raiz.setPai(raiz);
        int calc = 0;
        ArrayList<Vertice> vertices = new ArrayList<Vertice>(); // controlar o caminho percorrido
        ArrayList<Vertice> caminho = new ArrayList<Vertice>(); // caminho definido, se encontrar.
        Stack<Vertice> pilha = new Stack<>();
        pilha.add(raiz);
        raiz.setVisitado(true);

        while ( !pilha.isEmpty()) {

            Vertice verticeAtual = pilha.pop();
            System.out.print(verticeAtual.getNomeVertice()+" ");
            vertices.add(verticeAtual);
            if ( verticeAtual.getNomeVertice().equals(destino.getNomeVertice())) {

                String nome = verticeAtual.getPai().getNomeVertice();
                calc = verticeAtual.getCusto() + calc;
                caminho.add(verticeAtual);
                while( !nome.equals(raiz.getNomeVertice())) {

                    for ( int i = 0; i < vertices.size(); i++) {

                        if ( vertices.get(i).getNomeVertice().equals(nome)) {
                            //System.out.println("entrou no if do for do wwhile");
                            //System.out.println(vertices.get(i).getNomeVertice());

                                nome = vertices.get(i).getPai().getNomeVertice(); // atualiza quem Ã© o pai
                                caminho.add(vertices.get(i)); // guarda o vertice no caminho
                                calc = vertices.get(i).getCusto() + calc;
                        }
                    }
                }
                caminho.add(raiz);
                break;
            }

            for( int i = 0; i < verticeAtual.getVizinhos().size(); i++) {

                if ( !verticeAtual.getVizinhos().get(i).isVisitado()) {
                    verticeAtual.getVizinhos().get(i).setVisitado(true); // marca ja foi visitado
                    verticeAtual.getVizinhos().get(i).setPai(verticeAtual); // setar nome do pai
                    verticeAtual.getVizinhos().get(i).setCusto(verticeAtual.getPesosVizinhos().get(i)); // custo
                    pilha.push(verticeAtual.getVizinhos().get(i));
                }
            }
        }
        System.out.println("");
        System.out.println("Caminho solucao = ");
        for ( int indice = caminho.size()-1; indice >= 0; indice--) {
            System.out.print(caminho.get(indice).getNomeVertice()+" ");
        }
        System.out.println("Custo = "+calc);
    }
}
```

Fonte: Autoria própria

Figura 23 - Conjunto de teste A - Resultado

JPLAG	Metodologia
90-100(%)	89.7%

Fonte: Autoria própria

Figura 24 - Conjunto de teste B - Código 1

```
#include <stdio.h>

int main(void) {

        int inteiro = 0;
        printf("Valor da var inteiro antes: %d\n", inteiro);
        float real = 1.1;
        printf("Valor da var real antes: %f\n", real);
        char caracter = 'a';
        printf("Valor da var char antes: %c\n", caracter);
        int *pInteiro;
        float *pReal;
        char *pCaracter;
        pInteiro = &inteiro;
        pReal = &real;
        pCaracter = &caracter;
        *pInteiro = 1;
        printf("Valor da var inteiro depois: %d\n", inteiro);
        *pReal = 2.2;
        printf("Valor da var inteiro antes: %f\n", real);
        *pCaracter = 'b';
        printf("Valor da var inteiro antes: %c\n", caracter);

}
```

Fonte: Autoria própria

Figura 25 - Conjunto de teste B - Código 2

```
#include <stdlib.h>
#include <stdio.h>

int func1 (char a, int b, float c);

int main(void) {

        int (*ponteiro) (char, int, float);
        ponteiro = func1;
        char x = 'z';
        int y = 1;
        float z = 2;
        int inteiro = ponteiro(x,y,z);   // int inteiro = func1(.....) // (*ponteiro)(....);
        printf("valor do retorno func %i", inteiro);

}
int func1 (char x, int a, float h) {
        printf("\nnadaaa aqui\n");
        int s = a;
        return s;
}
```

Fonte: Autoria própria

Figura 26 - Conjunto de teste B - Resultado

JPLAG	Metodologia
0-10(%)	52.5%

Fonte: Autoria própria

Figura 27 - Conjunto de teste C - Código 1

```c
#include <stdlib.h>
#include <stdio.h>

int main(void) {

        int *vetor = NULL;
        int n;
        printf("\nDigite o tamanho do vetor = ");
        scanf("%d", &n);
        int elemento;
        vetor = (int *) malloc (n * sizeof (int));
        int i;
        int soma = 0;
        for ( i = 0; i < n; i++) {
                printf("\nDigite um elemento para a posicao %d do vetor = ", i);
                scanf("%d", &elemento);
                vetor[i] = elemento;
        }
        // Imprimir o vetor
        for ( i = 0; i < n; i++) {
                printf("\nVetor[%d] = %d", i, vetor[i]);
                if ( vetor[i] % 2 != 0) {
                        soma = soma + vetor[i];
                }
        }
        // A soma dos impares
        printf("\nA soma dos impares = %d", soma);
        free(vetor);
        return 0;
}
```

Fonte: Autoria própria

Figura 28 - Conjunto de teste C - Código 2

```c
#include <stdlib.h>
#include <stdio.h>

typedef struct {

        int carbonos;
        int hidrogenios;
}Hidrocarboneto;

int main(void) {

        Hidrocarboneto *h = NULL;
        h = (Hidrocarboneto *) malloc (sizeof(Hidrocarboneto));
        int c, i;
        printf("\nDigite o numero de carbonos = ");
        scanf("%d", &c);
        h->carbonos = c;
        printf("\nDigite o numero de hidrogenios = ");
        scanf("%d", &i);
        h->hidrogenios = i;
        int massa = 0;
        massa = (h->carbonos * 12) + (h->hidrogenios * 1);
        printf("\nMassa atomica do hidrocarboneto analisado = %d", massa);
        free(h);
        return 0;
}
```

Fonte: Autoria própria

Figura 29 - Conjunto de teste C - Resultado

JPLAG	Metodologia
0-10(%)	54.7%

Fonte: Autoria própria