UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA

RODRIGO ELIAS FRANCISCO

Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte

Goiânia
2016

TERMO DE CIÊNCIA E DE AUTORIZAÇÃO PARA DISPONIBILIZAR AS TESES E DISSERTAÇÕES ELETRÔNICAS NA BIBLIOTECA DIGITAL DA UFG

Na qualidade de titular dos direitos de autor, autorizo a Universidade Federal de Goiás (UFG) a disponibilizar, gratuitamente, por meio da Biblioteca Digital de Teses e Dissertações (BDTD/UFG), regulamentada pela Resolução CEPEC nº 832/2007, sem ressarcimento dos direitos autorais, de acordo com a Lei nº 9610/98, o documento conforme permissões assinaladas abaixo, para fins de leitura, impressão e/ou *download*, a título de divulgação da produção científica brasileira, a partir desta data.

1. Identificação do material bibliográfico: **[X] Dissertação** **[ ] Tese**

2. Identificação da Tese ou Dissertação

Nome completo do autor: Rodrigo Elias Francisco

Título do trabalho: Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte

3. Informações de acesso ao documento:

Concorda com a liberação total do documento [X] SIM [ ] NÃO[1]

Havendo concordância com a disponibilização eletrônica, torna-se imprescindível o envio do(s) arquivo(s) em formato digital PDF da tese ou dissertação.

Rodrigo Elias Francisco

Assinatura do (a) autor (a)

Data: 09 / 09 / 2016

[1] Neste caso o documento será embargado por até um ano a partir da data de defesa. A extensão deste prazo suscita justificativa junto à coordenação do curso. Os dados do documento não serão disponibilizados durante o período de embargo.

RODRIGO ELIAS FRANCISCO

Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte

Dissertação apresentada ao Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás, como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Área de concentração: Ciência da Computação.

Orientadora: Profa. Ana Paula Laboissière Ambrósio

Goiânia
2016

Ficha de identificação da obra elaborada pelo autor, através do
Programa de Geração Automática do Sistema de Bibliotecas da UFG.

Elias Francisco, Rodrigo
Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte [manuscrito] / Rodrigo Elias Francisco. - 2016.
CIX, 109 f.: il.

Orientador: Profa. Dra. Ana Paula Laboissière Ambrósio.
Dissertação (Mestrado) - Universidade Federal de Goiás, Instituto de Informática (INF), Programa de Pós-Graduação em Ciência da Computação, Goiânia, 2016.
Bibliografia. Apêndice.
Inclui siglas, gráfico, tabelas, algoritmos, lista de figuras, lista de tabelas.

1. Juiz Online. 2. Ensino de Programação. 3. Plágio. 4. Dificuldade de Problemas. I. Laboissière Ambrósio, Ana Paula, orient. II. Título.

CDU 004

MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE GOIÁS
INSTITUTO DE INFORMÁTICA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UFG

ATA Nº 18/2016

ATA DA SESSÃO DE JULGAMENTO DA DISSERTAÇÃO DE MESTRADO DE RODRIGO ELIAS FRANCISCO

Aos trinta dias do mês de agosto de dois mil e dezesseis, às catorze horas, na sala 150 do Instituto de Informática da Universidade Federal de Goiás, Campus Samambaia, reuniu-se a banca examinadora designada na forma regimental pela Coordenação do Curso para julgar a dissertação de mestrado intitulada **"Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte"**, apresentada pelo aluno Rodrigo Elias Francisco como parte dos requisitos necessários à obtenção do grau de Mestre em Ciência da Computação, área de concentração Ciência da Computação. A banca examinadora foi presidida pela orientadora do trabalho de dissertação, Professora Doutora Ana Paula Laboissière Ambrósio (INF/UFG), tendo como membros o Professor Doutor Humberto José Longo (INF/UFG) e o Professor Doutor Fernando Barbosa Matos (IFGoiano). Aberta a sessão, o candidato expôs seu trabalho. Em seguida, o aluno foi arguido pelos membros da banca e:

(X) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, sem restrições.

( ) tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **aprovação** do candidato, condicionado a satisfazer as exigências listadas na Folha de Modificação de Dissertação de Mestrado anexa à presente ata, no prazo máximo de 60 dias, a contar da presente data, ficando o professor orientador responsável por atestar o cumprimento dessas exigências.

( ) não tendo demonstrado suficiência de conhecimento e capacidade de sistematização do tema de sua dissertação, a banca concluiu pela **reprovação** do candidato.

Os trabalhos foram encerrados às dezessete horas. Nos termos do Regulamento Geral dos Cursos de Pós-Graduação desta Universidade, lavrou-se a presente ata que, lida e julgada conforme, segue assinada pelos membros da banca examinadora.

Profa. Dra. Ana Paula Laboissière Ambrósio _____

Prof. Dr. Humberto José Longo _____

Prof. Dr. Fernando Barbosa Matos _____

UNIVERSIDADE FEDERAL DE GOIÁS

INSTITUTO DE INFORMÁTICA

AUTORIZAÇÃO PARA PUBLICAÇÃO DE DISSERTAÇÃO EM FORMATO ELETRÔNICO

Na qualidade de titular dos direitos de autor, **AUTORIZO** o Instituto de Informática da Universidade Federal de Goiás – UFG a reproduzir, inclusive em outro formato ou mídia e através de armazenamento permanente ou temporário, bem como a publicar na rede mundial de computadores (*Internet*) e na biblioteca virtual da UFG, entendendo-se os termos "reproduzir" e "publicar" conforme definições dos incisos VI e I, respectivamente, do artigo 5º da Lei nº 9610/98 de 10/02/1998, a obra abaixo especificada, sem que me seja devido pagamento a título de direitos autorais, desde que a reprodução e/ou publicação tenham a finalidade exclusiva de uso por quem a consulta, e a título de divulgação da produção acadêmica gerada pela Universidade, a partir desta data.

Título: Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte

Autor(a): Rodrigo Elias Francisco

Goiânia, 30 de Agosto de 2016.

_____

Rodrigo Elias Francisco – Autor

_____

– Orientador

RODRIGO ELIAS FRANCISCO

Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte

Dissertação defendida no Programa de Pós–Graduação do Instituto de Informática da Universidade Federal de Goiás como requisito parcial para obtenção do título de Mestre em Ciência da Computação, aprovada em 30 de Agosto de 2016, pela Banca Examinadora constituída pelos professores:

Profa. Ana Paula Laboissière Ambrósio
Instituto de Informática – UFG
Presidente da Banca

Prof. Humberto José Longo
INF – UFG

Prof. Fernando Barbosa Matos
IFGoiano

Rodrigo Elias Francisco

Graduou-se em Sistemas de Informação pela Faculdade de Caldas Novas (2007) e especializou-se em Segurança da Informação pela Universidade Federal de Goiás - Campus Catalão (2011). Atuou como analista de sistemas nas áreas de Gestão Pública, Telecomunicações e Turismo. Trabalha como professor do Instituto Federal Goiano - Campus Morrinhos com disciplinas nas áreas de Programação de Computadores e Engenharia de Software. Atualmente, pesquisa ferramentas para facilitar a aprendizagem de programação de computadores.

Dedico este trabalho à minha mãe, que sempre me incentivou a estudar.

Agradecimentos

Primeiramente a Deus.

Aos meus pais, Nelza e Luiz, que fizeram de tudo para que eu seguisse meus estudos por toda a vida.

À Laiz, por ser uma irmã de verdade e pelo indispensável apoio dado a partir de uma visão psicológica e política.

À minha namorada, Paula, por ser paciente comigo nos momentos em que estive ausente, por alegrar os meus dias e por ser parceira em tudo.

À minha vó materna, Benedita, que encara todas as dificuldades impostas pela vida e passa isso como uma virtude para toda a família.

À minha tia Dicléia, que sendo tia-pedagoga sempre fez parte de minha educação. Lembro como se fosse hoje das aulas na varanda de sua casa, de escutar as mamonas estourarem com o calor do sol, e de brincar com os colegas na rua! Era muito legal.

À minha orientadora, Ana Paula L. Ambrósio, por ter me dado a oportunidade de crescer como estudante diante da rica experiência vivenciada e dos ensinamentos sobre pesquisa científica.

Às aulas do professor Humberto Longo, que juntamente com as conversas com a minha irmã e um olhar para toda a minha carreira profissional, me trouxeram questionamentos sobre como as pessoas aprendem algoritmos.

A todos os amigos que fiz no mestrado. Marcos Alves, Ernesto, Ana Lyvia, Daniela, Kátia, Heyde, Cleon, Porthos, Matheus, Caio, Fernando pelas boas conversas e risadas nos corredores do INF e pela oportunidade estudarmos juntos. Em especial ao Cleon pelas nossas conversas sobre a condução das pesquisas em Informática e Educação, que influenciaram positivamente para a qualidade deste trabalho.

Ao colegas do IFGoiano e alunos, que proporcionaram situações que incentivaram na busca deste conhecimento. Em especial ao Norton, Ana Maria e Cinthia, pelo grande incentivo e aos bons conselhos.

"todo conhecimento é inacabado, no sentido de que é um processo que se desenvolve continuamente, incorporando novos elementos e jamais deixando de questionar, sobretudo a si mesmo"

Paulo Freire,
Pedagogia da Autonomia.

Resumo

Francisco, Rodrigo Elias. **Juiz Online no Ensino de CS1: Requisitos, Dificuldade de Problemas e Plágio em Código-Fonte**. Goiânia, 2016. 109p. Dissertação de Mestrado. Instituto de Informática, Universidade Federal de Goiás.

Esta dissertação aborda Juiz Online no ensino de Programação Introdutória (CS1). Inicialmente houve uma pesquisa exploratória sobre o sistema BOCA no ensino de CS1, que trouxe experiências e dados de interações de alunos com o sistema, que, apoiados pela Revisão Sistemática da Literatura (RSL), contribuíram para a definição dos requisitos necessários para que o sistema atenda à disciplina de CS1 e guiaram a continuidade da pesquisa. Em um segundo momento, o objetivo foi resolver problemas específicos levantados na fase anterior, sendo a medição da dificuldade de problemas de CS1 e o apoio à identificação de plágio em atividades de CS1. A solução desses problemas contou com RSL, experiências práticas com escrita e execução de algoritmos, comparação dos resultados obtidos com os resultados esperados, e comparação das abordagens propostas com as identificadas na literatura. A estratégia proposta para medir a dificuldade de problemas de CS1 trabalha com a altura de uma árvore montada com conjuntos e sub-conjuntos de códigos aninhados num programa e a quantidade de assuntos relacionados. A estratégia para apoiar a identificação de plágio proposta trabalha com o algoritmo Distância de Edição no processamento e técnicas de normalização no pré-processamento. Trata-se de uma proposta fortemente adaptada à realidade dos dados utilizados nesta pesquisa (programas escritos em C, com poucas linhas de código, por alunos de CS1). A experiência mostrou a complexidade em aplicar a computação à educação, que trabalha frequentemente com dados subjetivos. Foi necessário levantar a dificuldade dos problemas na visão dos alunos e a visão de professores sobre a existência de plágio em pares de programas, cujas opiniões são bastante variáveis. Sugere-se para a evolução da área, que sejam criadas equipes multidisciplinares (com profissionais de computação, estatística, psicologia e pedagogia) e haja um foco na validação e no método usado para as pesquisas.

Palavras–chave

Juiz Online, Ensino de Programação, Plágio, Dificuldade de Problemas.

Abstract

Francisco, Rodrigo Elias. **Online Judge in CS1 Teaching: Requirements, Difficult Problems and Plagiarism in Source Code**. Goiânia, 2016. 109p. MSc. Dissertation. Instituto de Informática, Universidade Federal de Goiás.

This dissertation approaches Online Judge in teaching Introductory Programming (CS1). Initially there was an exploratory research on BOCA system in teaching CS1, which brought experiences and data of student interactions with the system, which, supported by Systematic Literature Review (RSL), contributed to defining the requirements for the system meets the discipline of CS1 and guided continuing research. In the second phase, there was the aim to solve specific problems identified in the previous phase, and measuring the difficulty of CS1 problems and support for plagiarism identification in CS1 activities. The solution of these problems included RSL, practical experiences with writing and execution algorithms, comparison of the results with the expected results, and comparison of the proposed approaches to the identified in the literature. The strategy to measure the difficulty of problems CS1 proposed works with the height of a tree mounted to sets and sub-sets of nested code into a program and the amount of related subjects. The strategy to support the identification of plagiarism proposal works with the Edit Distance algorithm processing and normalization techniques in preprocessing, and it is a highly adapted proposal to the reality of the data used in this research (programs written in C with few lines of code by students CS1). Experience has shown the complexity of applying computing to education, which often works with subjective data, it was necessary to raise the difficulty of the problems in view of the students and the teacher's view of the existence of plagiarism in peer programs, whose views are quite variables. It is suggested the creation of multidisciplinary teams to the evolution of the area (with professionals of computing, statistics, psychology and pedagogy) with a focus on validation and method used for research.

Keywords

Online Judge, Programming Education, Plagiarism, Problem Difficulty.

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Algoritmos

CAPÍTULO 1

Introdução

O mercado de Tecnologia da Informação e Comunicação (TIC) vem crescendo nos últimos anos no Brasil e não há mão de obra suficiente para atendê-lo [54]. Cursos como Ciência da Computação, Engenharia de *Software* e Sistemas de Informação formam profissionais para ocuparem as vagas nesse crescente mercado. Na base destes cursos estão as disciplinas de programação [22]. Porém, aprender a programar pode ser uma tarefa difícil.

A disciplina introdutória de programação, referenciada na literatura como CS1 (*Computer Science 1*), é fundamental para estes cursos. Porém esta disciplina apresenta diversos desafios e é responsável por altos índices de reprovação e abandono, não só na disciplina, mas no curso como um todo. No intuito de melhorar a situação, pesquisadores espalhados por todo o mundo estão investigando questões que se relacionam com o ensino de programação. O uso de diversas linguagens e metodologias têm sido propostos, mas o problema continua. De maneira geral, mais de um terço dos alunos de CS1 não passam na disciplina.

Martins e Mendes [42] destacam o relacionamento da disciplina de CS1 com processos cognitivos como a criatividade e racionalidade, que necessitam de meta-habilidades como abstração e inferência, e se relacionam com habilidades de leitura, interpretação e contextualização. Kramer [38] também destaca a necessidade da habilidade de abstração por parte dos alunos. Essa habilidade se desenvolve na fase de operações formais, explicada na teoria de Jean Piaget (1896-1980), e se inicia na adolescência, porém muitos adultos não conseguem pensar formalmente [34].

Além disso, nas empresas de *software*, o programador precisa lidar com situações em que a especificação detalhada do programa que precisa ser escrito tem poucos detalhes ou não existe. Mendonça et al. [44] propõem o uso de problemas mal definidos no ensino de CS1 com o objetivo de que o aluno gaste um tempo maior entendendo o problema antes de iniciar a codificação. A maioria dos alunos participantes da pesquisa iniciam a escrita do programa logo em seguida da leitura do problema e fazem apenas testes óbvios do programa.

Ambrósio et al. [11] também discutem sobre a frequente ocorrência de reprova-

ções e abandonos na disciplina CS1. Uma entrevista com professores sugere que entre os alunos que se desenvolvem com facilidade e os que aprendem pouco, estão os que têm dificuldade até certo ponto e depois se desenvolvem por meio de um "salto". O entendimento de que, primeiro, o aluno precisa pensar como um computador e passar para uma análise detalhada envolvendo o planejamento do algoritmo a partir do óbvio e global e, segundo, de que é preciso sair da visão concreta para a semântica e simbólica de maneira mais abstrata representa a opinião dos professores entrevistados. Muitos alunos negligenciam a fase de planejamento do algoritmo, passando da análise do problema diretamente para a escrita de código. A pesquisa infere que modelos mentais desenvolvidos em matemática e ciências ajudam nessa aprendizagem.

Dehnadi [24] apresenta a necessidade da formação de modelos mentais relacionados às estruturas utilizadas para resolver problemas de CS1, que se relacionam com a compreensão temporal das mudanças dos valores das variáveis e o entendimento da mecânica da programação, e Gomes et al. [31] explicam que a única forma de aprender a programar é programando. Logo, os alunos precisam realizar uma extensa prática de resolução de problemas com a escrita de programas e é necessário que exista *feedback* em tempo hábil. Este tipo de exercício desenvolve habilidades cognitivas, e permite criar modelos mentais importantes para o sucesso de programação.

Todas as questões apresentadas justificam a importância de propor metodologias adequadas para o processo de ensino-aprendizagem de CS1. Projetar metodologias viáveis depende da avaliação contínua das estratégias utilizadas considerando cada contexto de ensino. Dentre essas estratégias, o uso de ferramentas de Juiz *Online* auxilia o trabalho do professor e o torna mais produtivo ao facilitar a aplicação e correção de extensas listas de exercícios, trazendo a possibilidade de que o professor gaste seu tempo com questões didáticas.

1.1 Motivação

O desenvolvimento cognitivo do aluno necessita de prática. Nesse sentido, a aprendizagem de CS1 exige a resolução de muitos problemas e para isso os professores geralmente montam extensas listas de exercícios para os alunos resolverem. Porém, Chaves et al. [17] menciona que avaliar listas de exercícios em pouco tempo pode causar desgaste físico e mental nos professores e isso nem sempre é feito, prejudicando os alunos pela falta do *feedback* sobre as atividades que desenvolveram.

O *feedback* é fundamental para que o aluno consiga aprender CS1. Ele faz com que o aluno fique consciente do que não entendeu para que procure aprender. Ferramentas informatizadas associadas a um bom planejamento didático podem apoiar o trabalho do professor, reduzindo o desgaste físico e mental, e oferecer *feedback* aos alunos.

As ferramentas de Juiz *Online*, que disponibilizam problemas para os alunos resolverem através de programas em linguagens de programação pré-definidas e os corrigem automaticamente, auxiliam nesse processo. Alguns exemplos são: URI *Online Judge*, SPOJ Brasil, UVa *Online Judge* e ACM-ICP *Live Archive* [1, 3, 4, 5].

Esses sistemas geralmente possuem um repositório de problemas de programação classificados por tema e dificuldade. Alguns sistemas permitem ao professor montar listas a partir dos problemas do repositório, como o módulo acadêmico do *URI Online Judge* [4], outros somente disponibilizam os problemas para serem resolvidos em uma página *web*, sem ser possível nenhuma intervenção externa, como o SPOJ Brasil [3]. Essas características limitam a opção do professor na definição dos exercícios e no acompanhamento do aluno.

O Sistema *BOCA* [23], usado no evento Maratona de Programação organizado pela SBC (Sociedade Brasileira de Computação) e disponível como *software* livre para que instituições possam baixar e instalar, permite que o professor cadastre problemas e acompanhe seus alunos. Porém ele não é de fácil manipulação e não tem um banco de dados (BD) de questões para facilitar a geração de listas de problemas.

Apesar dos Juízes *Online* terem sido projetados para competições de programação, seus benefícios, como o *feedback* automático ao aluno e produtividade no trabalho do professor, proporcionam relevância em usá-los no ensino de CS1. A exploração de um Juiz *Online* no ensino de CS1 possibilita identificar características que mostrem as vantagens, desvantagens e informações necessárias à derivação de requisitos que visem uma melhor adaptação do sistema ao contexto de ensino. Outra questão importante é o benefício ao guardar dados de interações entre alunos e problemas de CS1 registradas no juiz *online*. Essas interações possuem diversos dados relacionados ao aluno e ao problema (número de tentativas, acerto, código-fonte submetido, data, hora), possibilitando fazer diversas análises nos dados, como comparar a similaridade nos códigos-fonte com a finalidade de analisar as possibilidades de plágio.

1.2 Objetivos

Este projeto segue uma abordagem de pesquisa exploratória apoiada por RSL (Revisão Sistemática da Literatura). Houveram dois momentos que definiram os objetivos. No primeiro momento, o objetivo foi explorar o sistema *BOCA* no ensino de CS1 e levantar os requisitos necessários de um sistema gestor de listas de exercícios. A intenção foi usar o sistema *BOCA* visando facilitar o trabalho do professor na geração de listas de problemas para alunos e o gerenciamento da correção das respostas. Dentre os requisitos levantados, escolher os mais relevantes a serem tratados de forma mais aprofundada.

Após levantamento inicial dos requisitos, foram selecionados a elaboração de uma estratégia para medir a dificuldade dos problemas de CS1 e outra para apoiar a identificação de plágio em código-fonte de atividades de CS1 como temas para serem tratados de maneira mais aprofundada.

De uma maneira mais específica, os objetivos são listados.

- Levantar os requisitos necessários para adaptar sistemas de Juiz *Online* para a elaboração automática de listas de exercício voltadas ao ensino de CS1;
- Propor uma estratégia para medir a dificuldade de problemas de CS1 com a finalidade de contribuir para a geração automática de listas de problemas;
- Propor uma estratégia para apoiar a identificação de plágio em código-fonte de atividades de CS1;

1.3 Metodologia

- **Pesquisa exploratória**
  Wazlawick [67] vê a pesquisa exploratória como o passo inicial ao estudar um fenômeno na busca de conhecer suas anomalias que visam oferecer subsídios para pesquisas mais específicas. Na experiência desta pesquisa, houve a consideração de publicações sobre o tema e o uso da ferramenta *BOCA* no ensino de CS1. Isso trouxe conhecimentos para a tomada de decisão sobre a condução do trabalho.
  Em um primeiro momento, ocorreram: (1) a criação de 100 problemas de CS1 no *BOCA*, (2) o entendimento dos requisitos do sistema *BOCA*, (3) modificações e implementações no sistema *BOCA* para a condução da pesquisa, e (4) uso da ferramenta *BOCA* com os alunos de CS1.
  Na sequência, houve um foco maior em estudar problemas específicos levantados pelo trabalho: (1) medição da dificuldade de problemas de CS1, Capítulo 3, e (2) estratégia para apoiar a identificação de plágio em atividades de CS1, Capítulo 4.
- **Fundamentação teórica**
  A leitura de trabalhos científicos, envolvendo desde artigos até teses de doutorado em língua inglesa e portuguesa, contribuiu para a fundamentação teórica. As RSLs realizadas trouxeram significantes contribuições para o desenvolvimento do trabalho.
  Houve o uso de trabalhos técnicos, como páginas da *web*. Esses trabalhos ofereceram boas contribuições para a pesquisa devido ao fato de explicarem de maneira mais enfática as ferramentas e algoritmos.
  A fundamentação teórica contribuiu para solucionar os problemas encontrados, oferecer uma visão abrangente dos assuntos explorados, e comparar as soluções dos

trabalhos estudados com as soluções propostas. Assim, associar a fundamentação teórica com a experiência prática foi de grande importância para a realização da pesquisa.

- **Implementação**

  Bezerra [14] explica que a fase de levantamento de requisitos tem o papel de tentar entender o domínio do negócio e compreende um estudo exploratório das necessidades de usuário e situação do sistema atual. A prática de uso do sistema *BOCA* no ensino de CS1 possibilitou a identificação desses requisitos.

  Houve também modificações em algumas partes específicas do Juiz *Online BOCA* para a condução deste trabalho. A manutenção é, de todas, a área da Engenharia de *Software* que consome a maior quantidade de trabalho, segundo Paduelli e Sanches [47]. A manutenção conta com a imprevisibilidade e isso dificulta a gestão e execução do trabalho. Essa etapa depende de entender a semântica do *software* a partir de seu código-fonte, realizar alterações, testar, avaliar impactos e implantar.

  O sistema *BOCA* não segue padrões arquiteturais e estruturais de *software*, como modelo-visão-controle e padrões de projeto. Este trabalho sugere a criação de um sistema que seja padronizado nesse sentido, para que possa evoluir com mais facilidade e melhore as oportunidades de comunicação e colaboração entre pesquisadores.

  As propostas para resolver a medição de dificuldade de problemas e o apoio à identificação de plágio contaram com o estudo e criação de algoritmos específicos. Foi feita uma abordagem para medir a dificuldade de problemas que envolveu o conceito de árvores e de pilhas no algoritmo construído [26]. Já o apoio à identificação de plágio contou com o estudo e uso do algoritmo Distância de Edição e técnicas de pré-processamento.

- **Escrita**

  Os resultados dessa pesquisa em âmbito geral constam nesta dissertação. Um artigo sobre a estratégia para medir a dificuldade dos problemas de CS1 no Juiz *Online BOCA* [26] foi publicado no *International Conference on Educational Data Mining - EDM 2015*. Um artigo sobre a estratégia para apoiar na identificação de plágio em código-fonte de atividades de CS1 [27] foi publicado na revista *iSys - Revista Brasileira de Sistemas de Informação*. Um artigo que aborda a RSL sobre Juiz *Online* e ensino de CS1 foi aceito para publicação no *XXVII Simpósio Brasileiro de Informática na Educação*, evento parte do *V Congresso Brasileiro de Informática na Educação*.

1.4 Organização da Dissertação

O Capítulo 2 busca responder como é a adaptação necessária para que um Juiz *Online* atenda à disciplina de CS1, usando uma abordagem que considera a experiência e a literatura. O Capítulo 3 aborda a dificuldade de problemas no contexto desta pesquisa. O Capítulo 4 aborda a identificação de plágio em atividades de CS1. Para validar as propostas dos capítulos 3 e 4 foram usados dados de interações de alunos com o sistema BOCA, obtida a partir da pesquisa exploratória realizada no Capítulo 3. O Capítulo 5 conclui o trabalho, apresentando os resultados, as discussões, a produção científica, e os trabalhos futuros.

CAPÍTULO 2

Sistemas de Juiz Online no ensino de CS1

Muitos alunos sentem dificuldades ao aprender CS1. Existem vários fatores que impactam na dificuldade dos alunos em aprender CS1. Gomes et al. [31] explicam que programar exige elevado nível de abstração, generalização e pensamento crítico. A fase inicial de aprendizagem necessita lidar com conceitos abstratos, como estruturas de controle, e de usar habilidades de resolução de problemas envolvendo raciocínio e lógica. Ao escrever programas, é necessário recordar regras sintáticas e características específicas da linguagem de programação juntamente com a construção de algoritmos. Do ponto de vista do método de estudo, a única forma de aprender a programar é programando. Logo, o estudo de CS1 é muito prático e intensivo.

A motivação exerce grande impacto na aprendizagem de CS1. Rountree et al. [55] aplicaram árvore de decisão para prever a aprovação ou reprovação de alunos na disciplina de CS1. Em todas as regras para a *aprovação* extraídas da árvore os alunos tem a expectativa de obter *A* na disciplina. Já as regras para a *reprovação* são feitas a partir da combinação de experiência acadêmica anterior à disciplina, conhecimentos matemáticos, idade, anos de estudo, e a falta da expectativa de obter *A* na disciplina. Esta questão da motivação também é apresentada por Gomes et al. [31].

Dehnadi [24] identificou que a dificuldade, de muitos alunos iniciantes de CS1, sobre a compreensão temporal das mudanças dos valores das variáveis atrapalha na compreensão da mecânica da programação. Existe também a ideia de que os modelos mentais desenvolvidos em matemática e ciências ajudam na aprendizagem de programação [20, 16, 68]. Estes problemas referem-se às dificuldades enfrentadas pelos alunos.

Existem outros fatores que se referem às dificuldades encontradas pelos professores, aos quais precisam dominar o conteúdo e executar seu trabalho com a didática adequada. Estes fatores incluem: dificuldade em lidar com a baixa motivação de alunos; falta de formação ou experiência docente; falta de ferramentas e material didático adequado; falta de boa remuneração; falta de tempo para planejar aulas; falta de tempo para corrigir soluções e dar *feedback* para os alunos.

Com base nesses aspectos, vários estudos estão sendo feitos no sentido de buscar maneiras melhores para ensinar CS1 e com isso gerar impacto positivo na formação dos

profissionais. Planejar estratégias pedagógicas mais adequadas depende de uma visão deste contexto.

Uma dessas linhas de pesquisa foca o problema enfrentado pelo professor na elaboração e correção de listas de exercícios. Visto que a parte prática do ensino de programação é fundamental para a aprendizagem, o desenvolvimento de ferramentas que ajudam neste processo tornam o trabalho do professor mais produtivo e contribuem para redução dos fatores que impactam negativamente o aluno, como a falta de *feedback*.

Um sistema de Juiz *Online*, também chamado de sistema de avaliação automática, oferece problemas para serem resolvidos com a submissão de códigos-fonte em uma linguagem de programação e os corrigem automaticamente. Estes sistemas são muito usados em competições de programação, devido a funcionalidade de automatizarem o processo de correção dos programas dos competidores. Campos e Ferreira [23] apresentam o sistema de juiz *online BOCA* usado nas maratonas de programação organizadas pela Sociedade Brasileira da Computação.

Um juiz *online* trabalha com casos de teste. Cada caso de teste possui um conjunto de entradas e suas respectivas saídas. Para verificar se a resposta de um problema é uma solução correta, um programa passa por seu respectivo conjunto de casos de teste.

2.1 Revisão Sistemática da Literatura

A RSL sobre Sistemas de Juiz *Online* e ensino de CS1 foi feita para contribuir com o entendimento do estado da arte. As questões de pesquisa que seguem visam oferecer uma visão abrangente do assunto.

- **QP:** Qual é a especificação mais adequada para que um Juiz *Online* atenda à disciplina de CS1?

- **QP1:** Quais os benefícios ao usar Juiz *Online* no ensino de CS1?

- **QP2:** Quais os problemas ao usar Juiz *Online* no ensino de CS1?

- **QP3:** Quais requisitos funcionais um Juiz *Online* precisa atender para ser usado no ensino de CS1?

- **QP4:** Quais requisitos não-funcionais um Juiz *Online* precisa atender para ser usado no ensino de CS1?

A RSL seguiu as instruções elaboradas por Kitchenham [36] e analisou trabalhos publicados no período de 2010 a 2016, sendo que a busca de artigos nas bases de pesquisa foi realizada na data 14/02/2016. Vieira [63], com a Figura 2.1, apresentou a estruturação dos passos da RSL usada nesta pesquisa.

Figura 2.1: *Fases de uma revisão sistemática da literatura - adaptado de [36] por [63].*

2.1.1 Planejamento

A fase de planejamento contou com a definição do protocolo, que é apresentado no Apêndice A. O protocolo apresenta as decisões técnicas sobre a condução da pesquisa, que inclui: as questões de pesquisa, características sobre a busca de artigos, *strings* de busca, critérios de inclusão, critérios de exclusão, e a especificação de como conduzir as fases de seleção e extração da RSL.

2.1.2 Execução

A fase de execução exige que sejam feitas a identificação dos trabalhos, a seleção e a extração. A seleção trouxe um total de 48 trabalhos, que se distribuem por base de pesquisa conforme a Figura 2.2 e pelo agrupamento de base de pesquisa e ano conforme Figura 2.3.

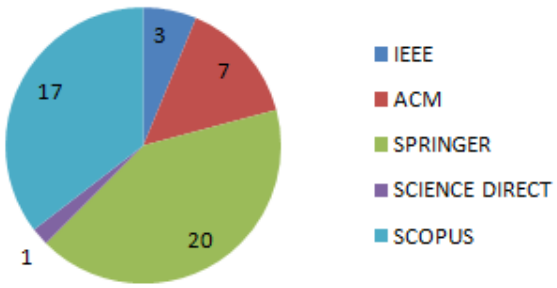

Figura 2.2: *Número de trabalhos identificados e agrupados por base de pesquisa.*

A Figura 2.2 mostra que as bases de pesquisas *Springer* e *Scopus* foram as que mais trouxeram resultados, com 20 e 17 trabalhos respectivamente.

A Figura 2.3 mostra que 2015 foi o ano em que houve o maior número de publicações, totalizando 13, e que durante os dois primeiros meses de 2016 incluidos na

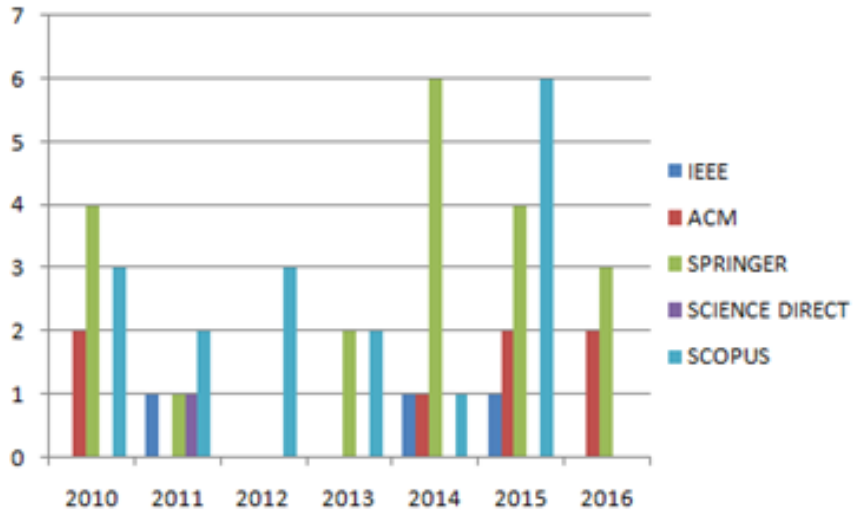

Figura 2.3: *Número de trabalhos identificados e agrupados por ano e base de pesquisa.*

pesquisa, de 01/01/2016 até 14/02/2016, ocorreram 5 publicações sobre o assunto. Isso indica um aumento no interesse de pesquisadores pelo assunto a partir de 2015.

A seleção contou com a leitura do título, palavras-chave e resumo de cada trabalho e uso dos critérios de inclusão e exclusão para definir os trabalhos que serão analisados por completo. A Figura 2.4 mostra os 36 trabalhos excluídos conforme critérios de exclusão: E1 - Texto completo não disponível para acesso gratuitamente na *web* ou no portal de periódicos da Capes; E3 - Trabalho não aborda Juiz *Online* e ensino de CS1; E5 - Publicação duplicada. Sobraram 12 artigos após execução dessa etapa.

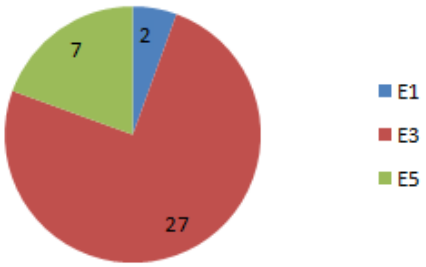

Figura 2.4: *Trabalhos removidos na seleção conforme critérios de exclusão*

A extração identifica os trabalhos que serão usados para responder às questões de pesquisa. Para isso, novamente foram usados os critérios de inclusão e exclusão, porém é feita a leitura completa dos trabalhos. Os critérios de inclusão usados foram: I1 - Aborda os benefícios ao usar Juiz *Online* no ensino de CS1; I2 - Aborda os problemas ao usar Juiz *Online* no ensino de CS1; I3 - Aborda os requisitos funcionais que um Juiz *Online* precisa atender para ser usado no ensino de CS1; I4 - Aborda os requisitos não-funcionais que um Juiz *Online* precisa atender para ser usado no ensino de CS1. Nesta fase, todos

os 12 trabalhos foram incluídos. A Tabela 2.1 mostra como esses trabalhos se distribuem nos critérios de inclusão.

Tabela 2.1: *Trabalhos incluídos na extração conforme critérios de inclusão.*

CRITÉRIO	REFERÊNCIAS	Nº TRABALHOS
I1	[66] [9] [49] [53] [40] [29] [48] [60] [64] [61]	10
I2	[66] [9] [50] [49] [40] [29] [51] [48] [60] [64] [61]	11
I3	[66] [9] [50] [53] [40] [29] [51] [48] [60] [64] [61]	11
I4	[53] [40] [29] [51] [48] [64] [61]	7

2.1.3 Conclusão

Após selecionar os artigos e ler cada um, foi possível sintetizar os dados e responder a cada uma das quatro questões de pesquisa, formando uma visão geral sobre o uso de Juízes *Online* no ensino de CS1.

QP1: Quais os benefícios ao usar Juiz *Online* no ensino de CS1?

Vários trabalhos destacam as vantagens do uso de sistemas de Juiz *Online*. Em Pettit et. al. [49], uma Revisão da Literatura concluiu que os professores pensam que as ferramentas melhoraram as suas experiências com o ensino, e que existem ferramentas que ajudam a melhorar a aprendizagem dos alunos, com *feedback* suficientemente preciso para ser útil. Souza et. al. [60] conclui em seu artigo, que a ferramenta ajuda os programadores novatos a aumentarem a qualidade de seus programas e conjuntos de testes. Vihavainen et. al. [64] afirmam que com o uso da ferramente *Test My Code*, houve um aumento de aprovação na disciplina, passando de 55% para 73%. Petit et. al. [48] conclui que devido a ferramenta *Jutge.org* oferecer uma lista de exercícios extensa, ela é útil para a aprendizagem de programação.

Mais pontualmente, a tabela 2.2 apresenta os benefícios encontrados nos artigos selecionados.

Um dos principais benefícios de um sistema de juiz *online*, é a facilidade de elaboração e correção de listas de exercícios de programação. Isto diminui a carga de trabalho do professor [66, 53, 9]. Além disto, a maioria dos sistemas tem um banco de questões associado, que permite a fácil elaboração de extensas listas de exercícios [48]. Visto que este banco de questões é desenvolvido com objetivos pedagógicos e com alto potencial de reuso, os problemas tendem a ser mais bem preparados e isso traz benefícios ao ensino [40, 61]. Muitas vezes os problemas vêm de livros usados em cursos introdutórios de programação [60].

Em diversos casos, os problemas são classificados de acordo com os conceitos envolvidos e o grau de dificuldade. No caso dos problemas criados no sistema *FLOP*, a

Tabela 2.2: *Benefícios encontrados nos trabalhos*

BENEFÍCIO	REFERÊNCIAS	Nº TRABALHOS
A - Existência de sistemas que são úteis para o ensino	[49, 60, 64, 48, 40, 61]	6
B - Redução da carga de trabalho do professor	[53, 66, 9]	3
C - Oferecer lista de exercícios extensa	[48]	1
D - Preocupações pedagógicas na criação dos problemas	[40, 61, 60]	3
E - Reduzir a ocorrência de plágio	[53]	1
F - Aprendizagem no ritmo do aluno	[40]	1
G - Auto-aprendizagem	[66]	1
H - Motivação	[29]	1
I - Melhorar a estratégia pedagógica	[53, 64]	2
J - Oferecer uma visão global e temporal do desenvolvimento do aluno	[9]	1
K - Assiduidade	[29]	1
L - Transparência	[29]	1

classificação levou em consideração os níveis básicos de domínio cognitivo da taxonomia de Bloom [40]. Esta classificação permite adaptar a escolha dos problemas ao nível de conhecimento do aluno.

Ferramentas de juiz *online* também podem ser usadas para automatizar a execução e correção de exames. Segundo Rajala et. al. [53], o ambiente controlado necessário para o uso da ferramenta *ViLLE* em exames contribui para uma baixa ocorrência de plágio, o que traz benefícios para pesquisas que necessitam de dados confiáveis para análise.

De fato, um outro importante benefício do uso de sistemas de juiz *online* é a possibilidade de armazenar as soluções submetidas para os problemas propostos. A análise destas soluções pode trazer informações relevantes para pesquisas em diversas áreas.

Do ponto de vista pedagógico, os sistemas de juiz *online* podem ser catalizadores de mudanças na forma de ensinar e aprender programação. A facilidade de obter *feedback* no momento que o aluno precisa permite que ele prossiga no seu ritmo [40], e promove a auto-aprendizagem [66] ao trazer aspectos da sintaxe, estrutura, semântica e resultados de testes no *feedback* dado. Isto pode ter um impacto na motivação dos alunos [29]. Professores também podem mudar o formato das aulas para incorporar o uso de sistemas de juiz *online* [53, 64].

A possibilidade de obter relatórios sobre o progresso individual de alunos e de turmas é um outro benefício do uso de sistemas de juiz *online* [9]. Estes oferecem uma visão global e temporal do desenvolvimento do aluno na disciplina.

A assiduidade e a transparência são outros benefícios educacionais apontados

no trabalho de Georgouli e Guerreiro [29]. A assiduidade é justificada pela facilidade do sistema controlar os prazos e a transparência se refere ao fato de que não há interferência de fatores subjetivos na correção dos problemas.

A Tabela 2.3 relaciona os sistemas de Juiz *Online* com os benefícios identificados. Os benefícios apresentados na Tabela 2.3 estão na forma de uma legenda (de A até L) cujos valores são descritos na Tabela 2.2. A marcação com * em **Experiência UL-FCIS* indica que nesse caso não é apresentado um sistema e sim uma experiência pelos autores.

Tabela 2.3: *Juízes Online relacionados aos benefícios.*

SISTEMA	A	B	C	D	E	F	G	H	I	J	K	L
Mooshak [29]								X			X	X
AutoLEP [66]		X					X					
eGrader [9]		X								X		
FLOP [40]	X			X		X						
Jutge.org [48]	X		X									
YOJ [61]	X			X								
Athene [50]												
**Experiência UL-FCIS* [51]												
ViLLE [53]		X			X				X			
ProgTest [60]	X			X								
Test My Code [64]	X								X			

QP2: Quais os problemas ao usar juiz online no ensino de CS1?

A Tabela 2.4 apresenta um resumo dos principais problemas identificados nos projetos referentes aos trabalhos lidos.

Tabela 2.4: *Problemas encontrados nos trabalhos.*

PROBLEMA	REFERÊNCIAS	Nº TRABALHOS
A - Falta de maturidade dos sistemas	[9, 51]	2
B - Falta de validação do sistema	[49, 40, 50, 51]	4
C - Dificuldade em gerenciar a infra-estrutura	[48]	1
D - Usabilidade	[60]	1
E - Disponibilizar publicamente a nota de todos os alunos	[29]	1
F - Sistema contextualizado em competições	[29, 61]	2
G - Falta da informação sobre a contribuição dos sistemas para a aprendizagem	[49, 29, 66]	3
H - Sistema aumentou ocorrência de plágio	[51]	1
I - Aluno não testar seus programas	[64]	1

Como pode ser verificado, grande parte dos problemas identificados estão relacionados à falta de maturidade dos sistemas e à dificuldade de instalação e manipulação

dos programas [9, 51]. A falta de validação dos sistemas de juiz *online* é um problema apontado em várias pesquisas [49, 40, 50, 51]. Existe também a preocupação com segurança e escalabilidade [48] que impacta na gestão da infra-estrutura. Além disto, existem problemas de usabilidade [60]. Estudantes, quando questionados sobre o processo de submissão, responderam que era um pouco complicado na primeira tentativa. O *feedback* da ferramenta não é suficiente e não contribui para que alunos consigam corrigir alguns erros específicos de programação.

Outra fonte importante de problemas é a percepção dos alunos, e como estes interagem com os sistemas de juiz *online*. Por exemplo, podem haver problemas entre os alunos quando as notas deles são visíveis para todos [29]. Também, alunos podem não gostar de sistemas com contextos ligados a competições. O vocabulário de concursos e *ranking* torna os alunos mais competitivos [29], o que pode ser encarado de maneira negativa. Por outro lado, Sun et. al. [61] afirma que as competições no ensino de CS1 são úteis para que alunos melhorem as habilidades de programação. Opiniões divergentes indicam que o problema deve ser investigado mais a fundo, pois está ocorrendo com frequência o uso da Gamificação em tecnologias educacionais.

A questão de como ou quanto os sistemas de juiz *online* contribuem na melhoria dos resultados dos alunos ainda é uma questão em aberto. Do ponto de vista dos alunos, ainda não existe uma visão clara sobre a contribuição do uso destes sistemas no seu aprendizado [49, 29]. Além disto, não parece haver aumento das notas dos alunos, com o uso destes sistemas [66].

Outro problema importante é a questão de plágio. Apesar deste não ser um problema exclusivo dos sistemas de juiz *online*, o fato das atividades muitas vezes serem feitas em casa [51], seu peso na nota e a grande quantidade de exercícios influenciam no aumento da ocorrência de plágio. No entanto, o fato da verificação ser automática permite a inclusão de verificadores de plágio nos sistemas.

Por fim, uso de ambientes de desenvolvimento, e o fato do sistema fazer a correção automática pode levar a uma programação na base de tentativa e erro. Os alunos acabam confiando nos testes apresentados pela ferramenta e não fazem seus próprios testes [64].

A Tabela 2.5 relaciona os sistemas de Juiz *Online* com os problemas identificados. Os problemas apresentados na Tabela 2.5 estão na forma de uma legenda (de A até I) cujos valores são descritos na Tabela 2.4.

QP3: Quais requisitos funcionais um juiz *online* precisa atender para ser usado no ensino de CS1?

A leitura dos trabalhos sobre CS1 e Juiz *Online* trouxe quatro conjuntos de requisitos funcionais. A Tabela 2.6 mostra a distribuição de trabalhos quanto a esses conjuntos.

Tabela 2.5: *Juízes Online relacionados aos problemas.*

SISTEMA	A	B	C	D	E	F	G	H	I
Mooshak [29]					X	X	X		
AutoLEP [66]							X		
eGrader [9]	X								
FLOP [40]		X							
Jutge.org [48]			X						
YOJ [61]						X			
Athene [50]		X							
**Experiência UL-FCIS* [51]	X	X						X	
ViLLE [53]									
ProgTest [60]				X					
Test My Code [64]									X

Tabela 2.6: *Requisitos funcionais encontrados nos trabalhos*

CONJUNTO DE REQUISITOS	REFERÊNCIAS	Nº TRABALHOS
C1 - *Feedback*	[29, 66, 9, 40] [48, 61, 50, 51] [53, 60, 64]	11
C2 - Integração do sistema com os cursos	[66, 40, 48, 61] [51, 53, 60, 64]	8
C3 - Análise do desempenho geral dos alunos	[48, 9, 64]	3
C4 - Oferecer diferentes tipos de atividades	[61, 51, 53, 60] [64]	5

Os requisitos necessários para que o sistema atenda à disciplina de CS1 dependem de cada contexto de ensino. Esta pesquisa identificou de uma maneira geral os requisitos. Respondendo a pergunta QP3, o ideal é um sistema que atenda aos requisitos identificados, mas que permita que o professor configure quais recursos deseja usar.

O *Feedback* é fundamental para a aprendizagem do aluno e é o requisito chave dos sistemas de *Juiz Online*. Os trabalhos apresentam maneiras diferentes de tratar o *feedback*.

Normalmente o *Feedback* é dado a partir dos resultados de testes feitos pelo Juiz *Online* com o programa submetido pelo aluno, o que é evidenciado nos trabalhos [40, 48, 51, 53, 61, 66]. Entretanto, Pettit et. al. [50] e Vihavainen [64] criaram estratégias para gerar *Feedback* com pontuação parcial a partir de resultados dos casos de testes. A possibilidade de professores darem *Feedback* foi tratada nos trabalhos [29, 61, 64]. A possibilidade de outros alunos darem *Feedback* foi apresentada no trabalho [61]. Já Rajala et. al. [53] projetaram um sistema que esconde o *Feedback* imediato devido ao uso em exames.

O uso das descrições dos erros de compilação e execução e da saída do programa compõe o *Feedback* proposto no trabalho [53]. Pettit et. al. [50] propõem um *Feedback*

que apresenta detalhamento dos casos de teste que produziram erro ao executar o programa submetido pelo aluno. Porém, Souza et. al. [60], na ferramenta *ProgTest*, criaram uma estratégia que avalia o programa do aluno com os casos de teste do professor e o programa que o professor escreveu para o problema com os casos de teste do aluno, além de permitir ao professor a inclusão de dicas textuais para serem apresentadas no *Feedback* relacionado a casos de teste definidos.

Wang et. al. [66], no seu trabalho sobre o Juiz *Online AutoLEP*, propõem um *Feedback* que considera aspectos de sintaxe e, com isto, é capaz de identificar pequenos casos de erros de sintaxe e lógicos. Houve também a produção de *Feedback* que consideram aspectos estruturais que se relacionam com a semântica [66, 9]. O Juiz *Online AutoLEP* consegue avaliar situações em que o aluno engana o sistema, como quando fazem um extenso programa que imprime diretamente os resultados sem usar estruturas indicadas, por meio de comparação de um grafo que representa o programa com modelos armazenados ligados à especificação do problema. O sistema *eGrader* verifica se o código-fonte corresponde à resposta correta por meio de semelhança estrutural considerando o grafo que representa o programa e métricas de *software*. Já Pettit et. al. [50], no sistema *Athene*, buscaram mostrar no *Feedback* a evolução do estilo de codificação usado pelo aluno no tempo, permitindo identificar os casos em que o aluno acerta 100% da questão com um programa escrito com código muito complexo.

Alguns trabalhos se preocuparam em integrar o sistema de Juiz *Online* com os cursos, trazendo novos requisitos.

Vihavainen et. al. [64] projetaram o Juiz *Online Test My Code* para uso em *MOOCs* com um viés pedagógico de aprendizagem extrema. Trata-se de um sistema, na forma de *plugin* para um *IDE*, que visou não sobrecarregar os alunos, se relacionar com a realidade de desenvolvimento de *software* de indústria, oferecer pontos parciais para exercícios, ser de fácil gestão por professores e aprimorar práticas de engenharia de *software* dos alunos. Já Petit et. al. [48] trabalharam no projeto do juiz *online Jutge.org*, com requisitos que consideraram ideias de sistemas de gestão de aprendizagem. Estas ideias possibilitam aos professores gerenciar cursos, adicionar problemas, anexar documentos, criar listas de problemas, monitorar o progresso dos alunos, interagir com alunos, gerenciar lista de alunos e tutores, criar trabalhos, criar concursos, criar exames, orientar alunos quanto às dificuldades nos problemas. A funcionalidade de gerir cursos foi tratada nos trabalhos [61, 60]. Sun et. al. [61] apresentam o sistema *YOJ* mostrando a possibilidade do professor configurar a página inicial do curso, gerenciar o material didático do curso, adicionar alunos ao curso, adicionar problemas para serem resolvidos em sala de aula e em casa, definir atributos relacionados aos problemas como dados para testá-los e definir a abordagem de teste. A possibilidade de professores cadastrarem novos problemas foi considerada nos trabalhos [40, 60].

Outras características com maior impacto aos alunos foram encontradas nos trabalhos. O uso dos sistemas em exames é apresentado nos trabalhos [66, 48, 53]. A detecção de plágio foi executada nos trabalhos [61, 51]. Pozenel et. al. [51] usaram o programa *MOSS* para detecção de plágio. A possibilidade de alunos entrarem no sistema sem cadastro foi considerada [40, 48]. Souza et. al. [60] têm como parte dos requisitos da ferramenta *ProgTest* oferecer a nota do aluno na solução de um problema e, para isto, realiza cálculos a partir de casos de teste do aluno contra o programa do aluno, casos de teste do instrutor contra o programa do aluno, e casos de teste do aluno contra o programa do instrutor. Houve o uso do sistema pelos alunos para resolver atividades para casa [61, 51]. Sun et. al. [61] criaram um fórum para discutir os problemas que proporcionou que os professores criassem regras para premiar os alunos que estão dispostos a ajudar os colegas respondendo às perguntas.

A classificação de problema por nível de dificuldade foi considerada [48, 53]. Rajala [53] propõem o cálculo do nível de dificuldade dos problemas usando pontuações individuais, contagens de submissão e tempo gasto em cada atividade. Petit et. al. [48] organizaram seus problemas também por tópicos.

Os trabalhos analisaram o desempenho geral dos alunos. A geração de relatórios com o desempenho de alunos específicos foi feita nos trabalhos [9, 48, 64]. Entretanto, AlShamsi e Elnagar [9] trabalharam com a geração de relatórios contendo o desempenho de turmas.

Problemas que exigem dos alunos sua resolução por meio da escrita de programas são os mais comuns nos Juízes *Online*. Porém, alguns tipos de atividades, diferentes das normalmente encontradas nos Juízes *Online*, foram identificadas nos trabalhos.

Problemas que possuem parte do código fonte escrito e solicitam ao aluno que resolva o restante foram propostos [61, 53]. Sun et. al. [61] entendem que isto permite avaliar do aluno conhecimentos específicos por obrigá-lo a seguir o desenvolvimento da forma que foi iniciado, facilitando a avaliação do entendimento de conceitos abstratos (e.g. matrizes, arquivos e recursividade). Pozenel et. al. [51] trabalharam com Orientação a Objetos, solicitando que alunos criassem classes com métodos que seguissem assinaturas pré-definidas para que pudessem ser testados. Por outro lado, Rajala et. al. [53] trabalharam com problemas que exigiam dos alunos o envio dos casos de teste para os programas feitos foram propostos pelo trabalho e Vihavainen et. al. [64], com a intenção de introduzir o conceito de Desenvolvimento Orientado a Testes, apresentaram para o aluno um conjunto de testes para serem usados usados localmente na solução dos problemas.

Pozenel et. al. [51] apresentaram problemas de CS1 envolvendo computação gráfica. Já Rajala et. al. [53] propõem problemas dos tipos (*Quiz*, *Code shuffling*, Exercício Robô e Ligue os Itens). Problemas do tipo *Quiz* possuem questões de múltipla escolha

e perguntas abertas curtas. Problemas do tipo *Exercício Robô* solicitavam que os alunos escrevessem um programa *Java* que controla um guindaste para mover caixas para uma posição solicitada. Problemas do tipo *Code shuffling* solicitavam que os alunos ordenassem as linhas de código embaralhadas que resolvem determinado problema. Problemas do tipo *Ligue os Itens* pediam aos alunos para arrastarem os itens na coluna direita aos locais corretos na coluna esquerda.

A Tabela 2.7 relaciona os sistemas de Juiz *Online* com os requisitos funcionais identificados que foram apresentados na Tabela 2.6. Os dados apresentados e sintetizados pela Tabela 2.7 mostram, de uma maneira geral, os requisitos considerados pelos sistemas e oferecem informações importantes para projeto de ferramentas de Juiz *Online* considerando experiências de pesquisas publicadas.

Tabela 2.7: *Juízes Online relacionados aos requisitos funcionais.*

SISTEMA	Feedback	Integração com os Cursos	Desempenho de Alunos	Diferentes Atividades
Mooshak [29]	X			
AutoLEP [66]	X	X		
eGrader [9]	X		X	
FLOP [40]	X	X		
Jutge.org [48]	X	X	X	
YOJ [61]	X	X		X
Athene [50]	X			
**Experiência UL-FCIS* [51]	X	X		X
ViLLE [53]	X	X		X
ProgTest [60]	X	X		X
Test My Code [64]	X	X	X	X

QP4: Quais requisitos não-funcionais um juiz *online* precisa atender para ser usado no ensino de CS1?

Os requisitos não-funcionais que o sistema precisa atender são dependentes dos requisitos funcionais definidos. A RSL levantou os requisitos não-funcionais que os trabalhos científicos apresentam.

A Tabela 2.8 mostra a distribuição dos requisitos não-funcionais encontrados nos trabalhos.

Tabela 2.8: *Requisitos não-funcionais encontrados nos trabalhos.*

REQUISITO	REFERÊNCIAS	Nº TRABALHOS
Integração	[64, 29]	2
Usabilidade	[40]	1
Segurança	[40, 48, 51, 53]	4
Escalabilidade	[48, 51]	2
Disponibilidade	[64, 40, 61]	3

A necessidade de integração entre sistemas de Juiz *Online* com outros sistemas é tratada nos trabalhos [64, 29]. Vihavainen et. al. [64] descrevem o sistema *Test My*

Code como um *Plugin* para ser ser integrado a um IDE padrão de indústria. Já Georgouli e Guerreiro [29] implementaram a integração do sistema de juiz *online Mooshak* com plataforma de *e-learning Caroline*.

A usabilidade é outro requisito não-funcional importante no projeto de sistemas para a educação. Llana et. al. [40] consideraram a usabilidade no projeto do sistema *FLOP*.

Existe um grande interesse em desenvolver sistemas de Juiz *Online* para serem amplamente usados na *internet*. Alguns trabalhos abordam a questão da segurança [40, 48, 51, 53]. No sistema *FLOP* [40], a avaliação das soluções enviadas pelos usuários é feita em um ambiente seguro e controlado para tentar impedir que programas mal-intencionados possam danificar o servidor. Existe limites no tempo de uso do CPU e no uso de memória pelos programas para evitar bloqueios do sistema, de forma que programas que excedem os limites estabelecidos são mortos. O tempo de execução de processos é controlado. Programas que não fazem nenhuma computação útil são mortos. Além disso, existe um controle no acesso pelos programas dos alunos ao sistema de arquivos e a recursos de rede. Pozenel et. al. [51] propõem a execução dos programas submetidos pelos alunos com um usuário sem privilégios no *Linux* para lidar com a segurança. O projeto *ViLLE*, usado em exames, lida com a segurança ao solicitar a supervisão da sala de aula, usar *firewall* para restringir o acesso à *internet* e permitir acesso a somente ao sistema e à *API Java* Rajala et. al. [53].

Diferente de outros trabalhos, Petit et. al. [48] descrevem que o sistema *Jude.org* executa todos os códigos-fonte enviados, que podem inclusive incluir códigos maliciosos. Para isso, foi necessário limitar o número de submissões permitidas e criar uma infra-estrutura com um sistema distribuído com seis servidores *Linux*. Nesta infraestrutura, o mestre, único que se conecta à internet, executa o Servidor Web, o Sistema Gerenciador de Banco de Dados e mantém a fila de submissões, enquanto os outros servidores, máquinas escravas criadas com máquinas virtuais, processam as filas de submissões e se conectam ao mestre a partir de uma rede privada.

A escalabilidade foi tratada pelos trabalhos [48, 51] e deve ser considerada quando se tem o interesse em um sistema com quantidade crescente de usuários e transações. A infra-estrutura proposta para o sistema *Jude.org* [48] faz com que o tempo de resposta do sistema seja quase real, em menos de 10 segundos para retornar *feedback*, tornando o sistema altamente escalável, i.e. sendo capaz de atender a uma crescente quantidade de usuários. Pozenel et. al. [51], para garantir a escalabilidade, consideram necessário que cada teste possuir um limite de 3 segundos para produzir uma saída, além de limitar o consumo de memória da Máquina Virtual *Java*.

O uso dos sistemas por uma grande quantidade de usuários traz a necessidade do requisito não-funcional disponibilidade, que pôde ser identificado nas descrições dos

sistemas apresentados nos trabalhos [64, 40, 61]. O fato do Juiz *Online Test My Code* ser usado em *MOOCs*, que geralmente disponibiliza cursos que atraem uma grande quantidade de alunos *online*, fez com que fosse necessário levar em consideração a questão da disponibilidade (Vihavainen et. al. [64]).

Llana et. al. [40] e Sun et. al. [61] apresentaram a necessidade do requisito não funcional disponibilidade ao propor sistemas que permitem o acesso por um número muito grande de alunos em qualquer lugar e momento.

A Tabela 2.9 relaciona os sistemas de Juiz *Online* com os requisitos não-funcionais identificados.

Tabela 2.9: *Juízes Online relacionados aos requisitos não-funcionais.*

SISTEMA	Integ.	Usabil.	Seg.	Escalab.	Disponib.
Mooshak [29]	X				
AutoLEP [66]					
eGrader [9]					
FLOP [40]		X	X		X
Jutge.org [48]			X	X	
YOJ [61]					X
Athene [50]					
**Experiência UL-FCIS* [51]			X	X	
ViLLE [53]			X		
ProgTest [60]					
Test My Code [64]	X				X

QP: Qual é a especificação mais adequada para que um juiz online atenda à disciplina de CS1?

Responder à essa questão requer a realização de pesquisas para levantar quais requisitos contribuem para a aprendizagem dos alunos nos diferentes contextos educacionais. A variedade possibilidades de requisitos funcionais e não-funcionais é grande.

Existe uma tendência nas pesquisas em buscarem modificar o *Feedback* oferecidos pelos sistemas aos alunos. Isso é exemplificado pelo trabalho de Pettit et. al. [50], que propõe Métricas de *software* para visualizar o estilo de codificação usado pelo aluno e identificar práticas de programação não saudáveis.

Os requisitos funcionais possíveis foram agrupados considerando sua semelhança, formando os conjuntos de requisitos: *feedback*, integração do sistema com os cursos, análise do desempenho geral dos alunos, oferecer diferentes tipos de atividades. Sun et. al. [61] acreditam que as competições no ensino de CS1 são úteis, porém, podem haver professores em determinados contextos educacionais que não se sentem confiantes em potencializar a competitividade nas turmas às quais ministram aulas. Enquanto a ferramenta *Jutge.org* [48] possui funcionalidades ligadas à gestão de cursos, o trabalho de

Georgouli e Guerreiro [29] integrou o Juiz *Online Mooshak* com outro sistema para usar a gestão de cursos. Isso demonstra a forte ligação dos requisitos com o contexto.

O requisito não-funcional Usabilidade gera influência sobre a aprendizagem do aluno. Questões relacionadas à segurança, disponibilidade e escalabilidade possibilitam expandir o número de usuários do sistema, característica necessária para trabalhos que visem um contexto maior.

Como as questões que dependem do contexto de ensino, apresentadas na RSL como Problemas e Benefícios, são importantes na definição dos requisitos, as mesmas foram tratadas nesta pesquisa. Pettit et. al. [49] expõem como inconclusivo o resultado da questão se os alunos pensam ou não que as ferramentas de avaliação automática de programas melhoraram os seus desempenhos e apresentam a falta de comparações formais para saber se as percepções negativas relacionam-se com o curso ou com as ferramentas. Relacionada a esta questão, a preocupação pedagógica na criação dos problemas foi identificada nos trabalhos [40, 61, 60].

Oferecer um ambiente favorável ao ensino exige que o professor invista tempo no trabalho e tenha informações verdadeiras sobre a realidade de sala de aula. A redução da carga de trabalho do professor foi apresentada nos trabalhos [53, 66, 9]. Contribuindo com isso, são propostas a possibilidade de acompanhar o progresso dos alunos no decorrer do tempo e a funcionalidade de detecção de plágio [61]. Isso dá indícios de que um ambiente bem preparado com estas ferramentas contribui para a aprendizagem e incentiva a continuação das pesquisas.

Por isso, entende-se que o grau de contribuição educacional das ferramentas é parte de uma série de outras características relacionadas ao contexto de ensino e que extraí-lo depende da realização de pesquisas envolvendo planejamento estatístico e comparação dos resultados. Após a identificação de muito esforço duplicado nas pesquisas sobre as ferramentas, é importante que haja colaboração entre Universidades e Países e que busquem uma métrica determinística compartilhada para medir a contribuição dessas ferramentas para a aprendizagem (Pettit et. al. [49]).

Isso sugere a criação de um sistema que possa ser configurado a cada contexto de ensino, contribuindo para o uso customizado e para a condução das pesquisas. A questão da Integração é importante para as pesquisas, pois pode haver Universidades em que o uso de determinado Ambiente Virtual de Aprendizagem já faz parte de sua maneira de trabalhar e que, mesmo assim, pretende usar um sistema de Juiz *Online* e também contribuir com as pesquisas.

2.2 Experiência com o BOCA no ensino de CS1

O sistema de Juiz *Online BOCA* foi usado em turmas de CS1 na Universidade Federal de Goiás e no Instituto Federal Goiano - Campus Morrinhos. Essa tarefa foi realizada com o objetivo de verificar quais são os benefícios e problemas encontrados no uso do sistema de Juiz *Online BOCA* na disciplina de CS1 e, a partir disso, tomar as decisões quanto à condução do trabalho.

2.2.1 Descrição do Juiz *Online BOCA*

O Juiz *Online BOCA* é usado para automatizar as competições de programação, como a Maratona de Programação organizada pela SBC, e, para isso, expõe problemas para serem resolvidos através de linguagens de programação e avalia as respostas submetidas (De Campos e Ferreira [23]). O domínio do sistema possui vocabulário contextualizado com Competição e Times. A Figura 2.5 apresenta um diagrama de caso de uso que contém as funções disponibilizadas pelo sistema *BOCA*.

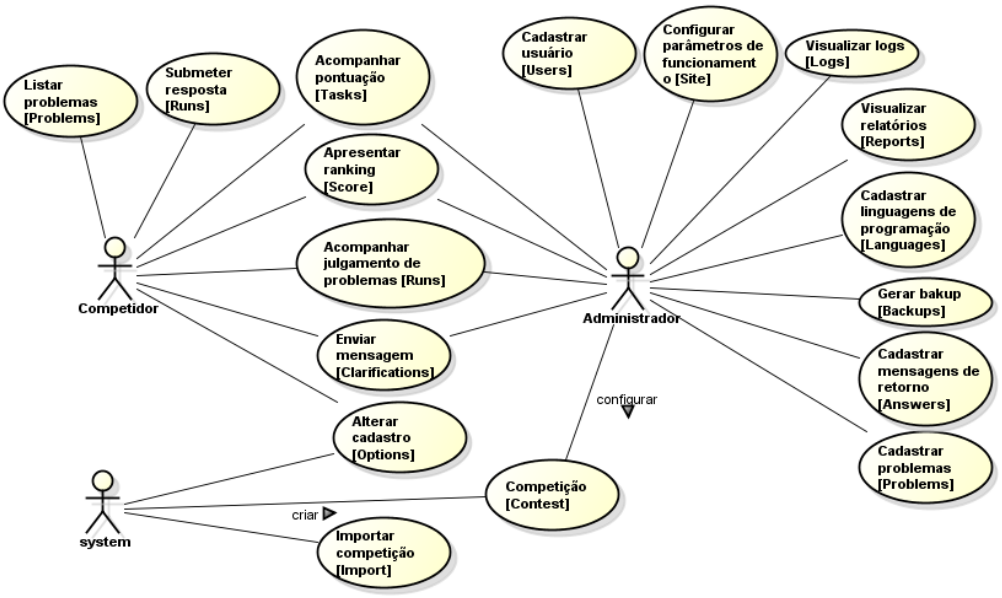

Figura 2.5: *Diagrama de Casos de Uso do BOCA.*

O sistema *BOCA* permite o cadastro de problemas para uma determinada competição. É preciso ter um arquivo PDF especificando o problema e os arquivos de entrada e saída para os casos de teste. Os problemas são associados a uma competição e para reutilizá-los em outra competição eles devem ser novamente inseridos. O *BOCA* não possui um banco de dados permanente de problemas.

Existem, no sistema, balões coloridos que representam pontos obtidos pelos alunos. Essa característica é parte do contexto das maratonas de programação, onde ao acertar um problema o time ganha um balão. Além de permitir ao time acompanhar as

submissões e os respectivos resultados, o *BOCA* também oferece uma tela que apresenta o *ranking* dos times contendo sua pontuação geral e uma lista dos exercícios resolvidos usando balões. A informação referente à interação dos alunos com os problemas contendo o tempo gasto e a quantidade de tentativas de solução também é apresentada.

O *BOCA* tem um *chat* que permite ao aluno fazer perguntas ao professor. O professor pode divulgar algum comentário global para os alunos pelo sistema. O professor pode analisar os programas enviados pelos alunos e definir no sistema qual resposta será submetida como *feedback*.

A interface de administração do sistema *BOCA* permite que sejam feitos as configurações de (competição, linguagens de programação e questões de prova), as inclusões de (times , juízes, placares, *staff* e administradores), e o controle das provas (início e parada). *Staff* é uma espécie de suporte, que auxilia desde a entrega de balões nas competições até a assistência a problemas de *hardware* e *software* e tem suas tarefas controladas pelo sistema. É possível realizar competições distribuídas com correção centralizada.

A preocupação com o requisito não-funcional Segurança foi verificada no sistema *BOCA*. Durante as competições, é indicado que os times não tenham acesso ao material externo ao computador e à internet. É disponibilizado aos times o acesso a ferramentas de desenvolvimento de *software*, como compiladores, editores de texto e ambientes integrado de desenvolvimento. Tudo o que o aluno faz em uma estação é salvo no servidor. Um sistema denominado *Maratona Linux* foi projetado para atender esse requisito.

Outras questões sobre segurança, relacionadas à infra-estrutura tecnológica, foram levantadas. O sistema é executado sobre o protocolo *HTTP* e permite execução sobre a versão segura *HTTPS*. O banco de dados, *PostgreSQL*, aceita autenticações seguras com autenticação *SSL*. É possível usar sistemas de *firewall* no sistema operacional.

A preocupação com a usabilidade do processo de instalação foi verificada. O sistema *Maratona Linux* foi projetado visando facilitar a instalação. Trata-se de um servidor, que não necessita de um especialista em administração de sistemas *Linux* durante a instalação, e máquinas cliente, com instalação ainda mais fácil.

O requisito não-funcional Disponibilidade pôde ser verificado no *BOCA* devido ao mesmo ser um sistema *web* escrito na linguagem *PHP* e que normalmente é executado com o servidor *Apache*. O resultado da avaliação do programa submetido pelo aluno, *feedback*, é transmitido o mais breve possível, relacionando-se com o requisito não-funcional Escalabilidade.

2.2.2 Análise da Experiência

A experiência de uso do sistema *BOCA* em turmas de CS1 trouxe benefícios ao facilitar a criação de listas de problemas através de competições com exercícios e oferecer um *feedback* automático para os alunos. Houve uma redução da carga sobre o professor, que fica como apoio para quando o aluno não consegue resolver os problemas. Isso permite que um maior número de exercícios possa ser feito e aumente a probabilidade de desenvolvimento das habilidades cognitivas necessárias para a programação.

Apesar das vantagens para o ensino, o uso da ferramenta no ensino de CS1 trouxe alguns problemas. Do ponto de vista da correção dos exercícios, o sistema, por corrigir os problemas através de uma comparação da saída da resposta com a saída esperada, exige rigor na formatação das respostas. Os alunos no início erram muito por não usarem a formatação correta, o que gera erro mesmo com a lógica adequada e causa uma certa frustração, já que o sistema não aponta onde está o erro. Porém, os alunos se adaptam e começam a prestar mais atenção com o tempo. O sistema permite que um aluno visualize o desempenho de outros, característica que precisa ser repensada considerando aspectos educacionais. Outro problema é que não existe um identificador de plágio das respostas. Observou-se que muitos alunos simplesmente copiam o programa do colega e o submete.

Apesar de De Campos e Ferreira [23] mencionarem a preocupação com a usabilidade do processo de instalação, foram encontradas características que contradizem isso. É apresentada a possibilidade de realizar competições distribuídas com correção centralizada, porém não conseguimos cadastrar várias competições e tê-las ativas em uma mesma instância do sistema. Ter uma instância do sistema para cada turma dificultou o reuso dos problemas. Além disso, o sistema não é trivial de gerenciar, o que acaba desestimulando o uso por parte de professores.

2.2.3 Requisitos de um sistema gestor de listas de exercícios de CS1

Diversas funcionalidades podem ser disponibilizadas em um Juiz *Online*, e estas normalmente se enquadram em *feedback*, integração do sistema com os cursos, análise do desempenho geral dos alunos, e oferecer diferentes tipos de atividades. A RSL disponibilizada neste capítulo oferece uma visão geral dessas funcionalidades, o que contribui para futuros projetos de Juiz *Online* ao permitir que sejam selecionadas quais características deseja implementar considerando a realidade de ensino. Em um contexto mais específico, resultado desta experiência, são apresentadas sugestões para a implementação de um sistema gestor de listas de exercícios usando o BOCA.

A Figura 2.6 apresenta diversas funcionalidades identificadas como relevantes para um sistema de gestão de listas de exercícios. O Caso de Uso R1, que visa cadastrar o aluno com informações detalhadas, foi proposto para contribuir com a criação de um

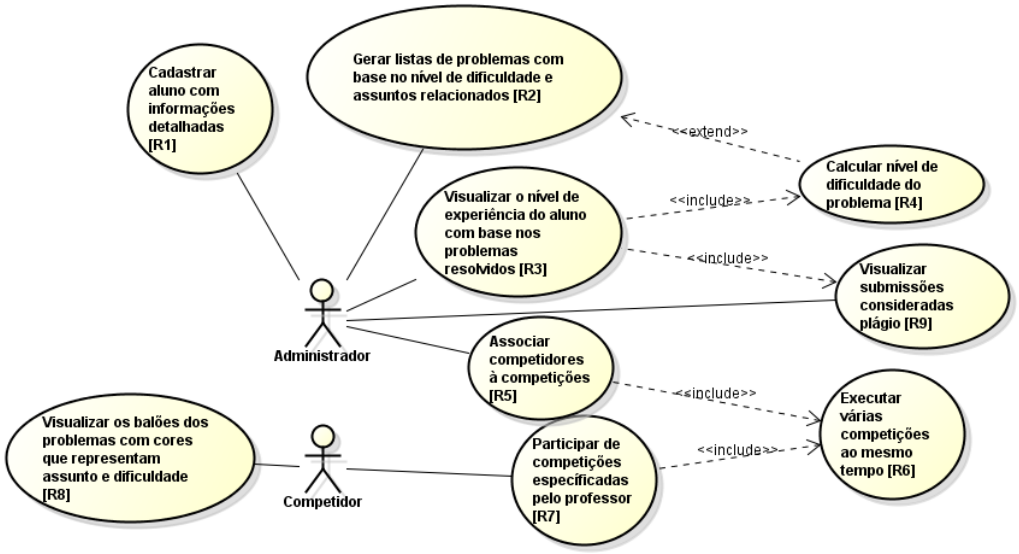

Figura 2.6: *Diagrama de Casos de Uso para atender aos novos requisitos.*

banco de dados de interações de alunos com os problemas, possibilitando a realização de trabalhos que busquem gerar conhecimento a partir desses dados. Outras funcionalidades propostas foram: gerar listas de problemas com base no nível de dificuldade e assuntos relacionados; visualizar o nível de experiência do aluno com base nos problemas envolvidos; calcular o nível de dificuldade do problema; associar competidores a competições; executar várias competições ao mesmo tempo; participar de competições especificadas pelo professor; visualizar os balões dos problemas com cores que representam assunto e dificuldade; e visualizar submissões consideradas plágio.

CAPÍTULO 3

Medição de dificuldade de problemas de CS1

A natureza da disciplina de CS1 proporciona aprendizagem por meio da resolução de problemas. Porém, é preciso ter um conjunto de problemas em níveis de dificuldade para apresentá-los aos alunos de maneira ordenada. Os requisitos de sistemas que apoiem o ensino, como Juízes *Online*, precisam considerar essas características.

Pesquisadores realizaram diferentes abordagens para resolver este problema. Alvarez e Scott [10] classificaram os níveis de dificuldade de problemas de CS1, na perspectiva de alunos novatos, a partir de dados sobre suas respostas. A análise estatística realizada levantou a existência de forte correlação do número total de linhas de código e número de fluxo de controle com a dificuldade do problema.

Mendonça et. al. [44] estudaram a dificuldade dos alunos de CS1 em lidar com a declaração do problema a partir da leitura e interpretação. A questão da abstração envolvida na declaração do problema e a dificuldade dos alunos em lidar com problemas mal definidos foi observada.

A funcionalidade de gerar listas de problemas no sistema BOCA foi identificada como importante para esta pesquisa, e depende de problemas com dificuldades classificadas. Por isso, este capítulo apresenta uma proposta para medir a dificuldade de problemas. Foram realizados experimentos embasados em outros trabalhos científicos. A seção 3.1 busca apresentar o cenário dessas pesquisas e oferecer uma visão do estado da arte.

3.1 Revisão Sistemática da Literatura

Com finalidade de entender o estado da arte sobre dificuldade de problemas de CS1, foi feita uma RSL. A intenção foi conhecer o estado da arte do tema em uma perspectiva abrangente, levando em consideração as questões de pesquisa que seguem.

- **QP:** Qual é a estratégia mais adequada para medir a dificuldade de problemas de CS1?

- **QP1:** Quais estratégias já foram usadas para medir a dificuldade de problemas de CS1?

- **QP2:** Como validar a estratégia para medir a dificuldade de problemas de CS1?

A RSL seguiu as instruções elaboradas por Kitchenham [36] e analisou trabalhos publicados no período de 2010 a 2016. A busca de trabalhos nas bases de pesquisa foi realizada na data 05/03/2016. A estruturação dos passos da RSL foi apresentada por Vieira [63] na Figura 2.1.

3.1.1 Planejamento

O protocolo consta no Apêndice C.

3.1.2 Execução

Foram identificados 35 trabalhos. Suas distribuições por base de pesquisa constam na Figura 3.1 e por base de pesquisa e ano constam na Figura 3.2.

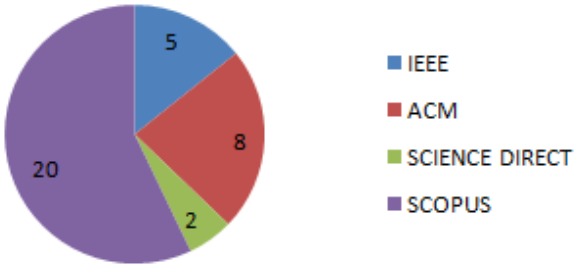

Figura 3.1: *Número de Trabalhos identificados agrupados por base de pesquisa.*

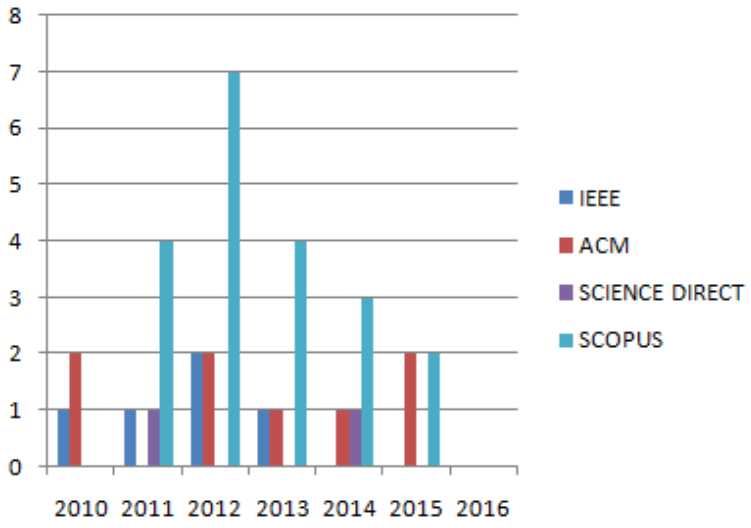

Figura 3.2: *Número de Trabalhos identificados agrupados por ano e base de pesquisa.*

A base de pesquisas *SCOPUS* foi a que mais trouxe resultados.

A seleção contou com a leitura do título, palavras-chave e resumo de cada trabalho e uso dos critérios de inclusão e exclusão. A Figura 3.3 mostra os 28 trabalhos excluídos conforme critérios de exclusão. Os critérios foram E3 - Trabalho não aborda dificuldade de problemas e CS1 e E5 - Publicação duplicada. Sobraram 7 artigos após execução dessa etapa.

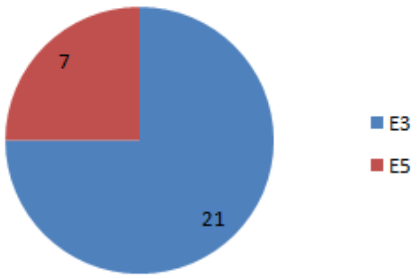

Figura 3.3: *Trabalhos removidos na seleção conforme critérios de exclusão.*

A extração identificou 6 trabalhos para responder às questões de pesquisa. Nessa fase houve a exclusão de um trabalho. O critério foi (E3: Trabalho não aborda dificuldade de problemas e CS1). A Tabela 3.1 mostra como esses trabalhos restantes se distribuem nos critérios de inclusão (I1 - Aborda estratégias usadas para medir dificuldade de problemas de CS1; I2 - Aborda validação de estratégias usadas para medir dificuldade de problemas de CS1).

Tabela 3.1: *Trabalhos incluídos na extração conforme critérios de inclusão.*

CRITÉRIO	REFERÊNCIAS	Nº TRABALHOS
I1	[15] [59] [40] [19] [25] [39]	6
I2	[15] [25]	2

3.1.3 Conclusão

A leitura dos trabalhos considerando os critérios da RSL permitiu realizar uma síntese dos dados. Isto possibilitou responder a cada uma das duas questões de pesquisa e formar uma visão geral sobre as pesquisas.

QP1: Quais estratégias já foram usadas para medir a dificuldade de problemas de CS1?

A Tabela 3.2 mostra os grupos de características encontrados nos trabalhos estudados.

A compreensão da dificuldade de problemas a partir dos erros ocorridos foi praticada nos trabalhos [15, 19, 39]. A partir de atendimentos de tutores a alunos

Tabela 3.2: *Características encontradas nos trabalhos sobre dificuldade de problemas de CS1*

CONJUNTO	CARACTERÍSTICA	REF.	Nº TRAB.
Erros	Erros dos alunos	[15, 39]	3
	Erros nos programas	[19]	
	Calculo de Percentuais de acertos	[19]	
Conceitos	Estrutura de repetição	[15] [19]	4
	Estrutura de seleção e Instruções switch	[19, 15]	
	Matrizes	[15]	
	Entrada e saída em arquivos	[15]	
	Strings	[15]	
	Sequência	[59]	
	Consideração de conceitos ensinados no curso até o momento da invenção da atividade	[25]	
Estratégias	Professor analisa a dificuldade	[40] [25]	6
	Análise da Solução escrita pelo autor do problema	[25]	
	Resolução do problema para estimar a dificuldade	[40]	
	Estuda a dificuldade para a disciplina de CS1 de uma maneira geral	[15] [59] [19]	
	Estuda a dificuldade específica do problema	[40] [25] [39]	
Resolução de Problemas	Raciocínio lógico	[15]	5
	Capacidade de síntese a partir da leitura do código-fonte	[59]	
	Dificuldade de lidar com o problema sem o computador	[39]	
	Compreensão textual	[25, 19, 15, 39]	

visando assistência em atividades, Bryce et. al. [15] categorizaram os principais erros dos estudantes com dificuldades na disciplina de CS1 ao escrever programas. Já Lakanen et. al. [39] investigaram os erros dos alunos na resolução do problema *Soloway's Rainfall* a partir de uma análise manual minuciosa e concluíram que os principais erros foram relacionados às tarefas (divisão por zero, contagem de entradas válidas, controle das repetições, cálculo da média envolvendo a sequência do algoritmo, e entendimento do texto).

A pesquisa de Cherenkova et. al. [19] sugere a tendência de certos tipos de erros nos programas que não são considerados pelas avaliações do sistema *CodeLab*, e.g. um problema que enfatiza estrutura de seleção possui o erro *Reversed comparison operator* executado por 28.52% dos estudantes. Por outro lado, Cherenkova et. al. [19] calcularam os percentuais de acertos dos problemas para identificar os problemas difíceis no sistema *CodeLab*.

Houve o interesse em associar a dificuldade dos problemas com os conceitos de CS1 nos trabalhos [19, 15, 59, 25].

Denny et. al. [25], ao medirem as dificuldades de problemas inventados pelos alunos, consideraram os conceitos que haviam sido ensinados no curso até o momento em que a atividade foi inventada. Relacionado a isso, Bryce et. al. [15] concluem que os conceitos (estrutura de repetição, instruções *switch*, matrizes, entrada/saída em arquivos, *strings*) se relacionam com as dificuldades dos alunos de CS1 e que estas dificuldades envolvem o entendimento da lógica e da sintaxe e Cherenkova et. al. [19] identificaram, a partir de uma base de 266.852 registros, que os conceitos de CS1 que os alunos têm mais

dificuldade são estrutura de seleção e estrutura de repetição.

No entanto, Simon [59] fez pesquisas que confirmaram a dificuldade dos alunos em entender a sequência dos algoritmos. Existem alunos que parecem acreditar que grupos de declarações são executados simultaneamente ao invés de sequencialmente e, por isso, não entendem atribuições de variáveis nos programas. Esse conceito é pré-requisito para que os alunos consigam entender os demais.

Diferentes estratégias para a condução das pesquisas foram identificadas nos trabalhos [15, 59, 19, 40, 25, 39].

Houve o interesse dos trabalhos [15, 19, 59] em buscar a dificuldade dos alunos na disciplina de CS1 como um todo. No entanto, a preocupação em levantar a dificuldade específica de cada problema foi verificada nos trabalhos [25, 39, 40]. Lakanen et. al. [39] investigaram a resolução do problema *Soloway's Rainfall*, que pede a criação de um programa que leia números inteiros até quando a entrada for igual a 99999 e escreva a sua média.

A prática de professores medirem as dificuldades dos problemas foi verificada nos trabalhos [40, 25]. Denny et. al. [25] mediram as dificuldades de problemas inventados pelos alunos a partir de análise humana envolvendo a descrição do exercício, a solução padrão escrita pelo autor e os conceitos que haviam sido ensinados no curso até o momento em que a atividade foi inventada. Já Llana et. al. [40], no projeto *FLOP*, usaram uma estratégia que solicita que professores resolvam os problemas para contribuir com a medição de sua dificuldade.

Segundo os trabalhos [15, 59, 39, 19, 25], as questões relacionadas à resolução de problemas impactam na dificuldades dos mesmos.

A capacidade de compreensão e o entendimento da lógica representam os maiores problemas relacionados às dificuldades dos alunos identificados na pesquisa de Bryce et. al. [15]. Lakanen et. al. [39] identificaram a dificuldade no entendimento do texto a partir da resolução do problema *Soloway's Rainfall*. Isso traz considerações importantes para a evolução da pesquisa.

Outros trabalhos realizados mostram a inclinação das pesquisas para essa linha que envolve a dificuldade de compreensão. Cherenkova et. al. [19] se empenharam para levantar a dificuldade textual do problema por meio de análises de comprimento do texto, vocabulário e referências culturais, porém os resultados não chegaram a nenhuma conclusão. Denny et. al. [25], para calcular a dificuldade de problemas de CS1, realizaram análises da descrição do exercício a partir da sua leitura.

Por outro lado, mas ainda envolvendo a resolução de problemas, Simon [59] menciona sobre a dificuldade dos alunos em ler um código-fonte e explicá-lo de maneira sintetizada e entende que essa dificuldade, em muitos casos, está relacionada à não compreensão da sequência do algoritmo.

QP2: Como validar a estratégia para medir a dificuldade de problemas de CS1?

A Tabela 3.3 resume as características encontradas nos trabalhos referente à validação.

Tabela 3.3: *Características encontradas nas validações dos trabalhos sobre dificuldade de problemas de CS1.*

CARACTERÍSTICA	REF.	Nº TRAB.
Impossibilidade de generalizar informações	[15]	1
Relacionar proporção de submissões corretas para cada problema com o nível de dificuldade	[25]	1

Existem ameaças que impedem generalizar os resultados da pesquisa de Bryce et. al. [15] a todos os estudantes. Essas ameaças apareceram ao observar que: (a) existem erros adicionais que os alunos não conhecem, (b) a coleta de dados ocorreu apenas de alunos que visitaram o laboratório de tutoria, e (c) a eficácia do professor pode distorcer esses resultados. É preciso considerar essas informações na validação das estratégias.

Denny et. al. [25] fizeram uma validação empírica para a classificação humana da dificuldade e a apontaram como satisfatória. A proporção de submissões corretas para cada problema foi usada, visando considerar que os exercícios mais fáceis possuem uma proporção maior de submissões bem sucedidas. Porém, a possibilidade de plágio pode ameaçar essa validação.

Outra informação importante, que recebe influência da validade do trabalho de Denny et. al. [25] mas não o valida, é a relação entre a dificuldade do exercício inventado pelo aluno e seu desempenho no exame. Essa relação mostrou que os estudantes que criaram exercícios mais difíceis tiveram melhor desempenho no exame.

QP: Qual é a estratégia mais adequada para medir a dificuldade de problemas de CS1?

Buscando responder a essa pergunta, sugere-se a realização de pesquisas com planejamento que considere as questões relacionadas à validação e que explorem as características relacionadas à dificuldade de problemas trazidas por esta RSL.

O estudo de Bryce et. al. [15] apresenta ameaças que impactam diretamente na validação desta pesquisa, e.g, a existência de erros adicionais que os alunos não conhecem, dados coletados apenas de alunos que visitaram o laboratório de tutoria e a eficácia do professor. Diferente da abordagem dos trabalhos [40, 25], que usam a classificação da dificuldade pelo professor, é sugerido que a validação trabalhe com dados que representem as opiniões dos alunos.

Denny et. al. [25] abordou o uso da informação de quais conceitos foram apresentados para os alunos até o momento da classificação da dificuldade, trazendo considerações importantes para a validação das pesquisas. Entende-se que a capacidade

do aluno é variável no tempo e existem alunos com diferentes capacidades em uma sala de aula. Logo, é importante trabalhar com conjuntos de alunos quantos às suas habilidades e/ou capacidades cognitivas para ter informações mais reais sobre a dificuldade.

Existem pesquisas que enfatizaram estudar a dificuldade de cada problema [40, 25, 39], que é o foco deste trabalho. Por outro lado, os trabalhos [15, 59, 19] estudaram de uma maneira geral a dificuldade dos conceitos da disciplina de CS1 e, mesmo assim, contribuem para esta pesquisa.

Quanto às características que relacionam-se com a dificuldade, é possível perceber uma inclinação em trabalhar com (1) conceitos envolvidos [15, 19, 59, 25], (2) aspectos referentes à resolução de problemas [15, 59, 39, 19, 25], envolvendo, e.g., a compreensão textual e a capacidade de síntese a partir da leitura do código-fonte, e (3) análise de erros [15, 39, 19].

Bryce et. al. [15] associaram a dificuldade de conceitos a erros ocorridos. Isso contribui para esta pesquisa devido ao fato de que os erros dos alunos se relacionam com as dificuldades impostas pelo problema. Simon [59] apresenta a necessidade do aluno compreender de forma sintetizada, ou relacional, o que código-fonte faz e relaciona isso com entendimento da sequencia do algoritmo, o qual, segundo o mesmo autor, muitos alunos de CS1 não possuem. Relacionado a isso, do mesmo modo que é preciso conhecer a sintaxe e a semântica da programação, é preciso que entender o problema em questão (Bryce et. al. [15]), relacionado-se com as habilidades de interpretação e resolução de problemas. A RSL apresenta uma variedade de pesquisas que podem ser feitas para resolver o problema em questão.

3.2 Estratégia Proposta

Para propor uma estratégia para medir a dificuldade de problemas de CS1, foram realizados experimentos com o objetivo verificar a relação entre a dificuldade, na visão dos alunos, com características de problemas criados para uso no sistema *BOCA*.

O levantamento de características dos problemas, planejado a partir da literatura, contou com: (1) o número de estruturas de repetição utilizadas no programa, conforme os trabalhos [10, 15, 19]; o número de estruturas de seleção, conforme os trabalhos [10, 19]; o número de linhas de código, conforme o trabalho [10]; o número de assuntos abordados no problema, que se relaciona com abordagens encontradas nos trabalhos [15, 19, 59, 25]; e a representação do código-fonte do programa em estrutura de dados na forma de árvore e grafo.

Os trabalhos [15, 19, 25, 39, 44, 59] discutiram a dificuldade relacionada ao entendimento e resolução do problema. Considerar o fato de que o algoritmo representa semanticamente a solução do problema e, para isso, usa regras formalizadas em uma

sintaxe permitiu identificar a oportunidade de representar a estrutura do programa na forma de grafo para estudar sua correlação com a dificuldade. Explorar a representação por grafo permitiu perceber de maneira empírica que a configuração de aninhamentos no código-fonte pode se relacionar com a dificuldade. Por isso, houve o interesse de trabalhar também com a representação por árvore.

Houve o uso de: número de arestas existentes em um grafo que representa a sequência do algoritmo, número de arestas que representam estruturas de repetição do grafo/sequência do algoritmo, e altura de uma árvore em que cada sub-grupo de códigos aninhados internos refere-se a um nó.

Este trabalho considera as dificuldades segundo o ponto de vista do aluno. Foram feitas comparações que mostraram a existência de uma diferença de pontos de vista entre alunos e professores para o conjunto de dados.

3.2.1 Coleta de Dados

Para explorar o uso do sistema *BOCA* no ensino de CS1, houve, no ano de 2014, a criação de 74 problemas para uso em sala de aula. Nesta etapa do projeto, cada problema foi classificado quanto ao assuntos de CS1 que aborda.

A classificação dos assuntos foi feita usando os seguintes tópicos: entrada de dados; saída de dados; estruturas de seleção; estruturas de repetição; cadeia de caracteres (*string*); estruturas de dados (vetores); uso de funções pré-definidas; definição de funções; passagem de parâmetros; domínio de lógica de programação; estrutura de dados (matriz); matemática.

Observou-se importante colher as opiniões dos alunos sobre a dificuldade dos problemas para realizar pesquisas. Para isso, o sistema *BOCA* foi alterado incluindo um campo de dificuldade do problema na tela de submissão de respostas a problemas. Com isso, a tabela do BD que guarda as submissões passou a armazenar também a dificuldade percebida pelo aluno. A dificuldade foi definida de um a cinco.

O banco de dados do sistema *BOCA* armazena dados dos alunos, dos problemas e do relacionamento entre alunos e problemas, representado pela tabela *runtable*. Esse relacionamento é de muitos-para-muitos e cada linha da tabela possui a identificação do aluno, do problema e os dados da atual submissão. Entre esses dados existe o código-fonte da resolução do problema em uma linguagem de programação, e a resposta gerada pelo Juiz *Online* informando se o usuário acertou o problema ou obteve algum erro.

Ocorreu a criação de uma competição no sistema *BOCA* para ser usada simulando uma lista de problemas nas disciplinas de CS1. Isto trouxe dados das dificuldades dos problemas na visão dos alunos, que proporcionaram a realização deste trabalho.

Houve também o estudo da correlação entre das variáveis com as dificuldades representando a percepção dos professores. Para isso, quatro professores com experiência na disciplina de CS1 no ensino Técnico e Superior leram os enunciados dos 74 problemas e definiram seus níveis de dificuldade, de um a cinco. Esse grupo foi formado por três professores e uma ex-professora de CS1 do IFGoiano (Instituto Federal Goiano).

3.2.2 Processamento

Após usar o sistema *BOCA* com os alunos nas turmas de CS1, decidiu-se analisar os dados das interações dos alunos com os problemas para entendermos melhor a dificuldade dos problemas.

A estratégia adotada neste trabalho foi a análise de correlações das variáveis dos problemas com as dificuldades definidas pelos alunos e pelos professores. A correlação entre as dificuldades definidas pelos alunos com as definidas pelos professores também foi analisada.

Este trabalho pesquisou variáveis associadas aos problemas de CS1 que se correlacionam com as dificuldades dos respectivos problemas. Para isso, houve a realização de atividades baseadas nas ideias de que: (1) pode ser interessante representar o algoritmo que resolve o problema por meio de grafo e árvores; e (2) a configuração dos assuntos de CS1 pode se correlacionar com a dificuldade do problema.

Houve o uso de um grafo que modela a sequência do algoritmo e de uma árvore com finalidade de verificar os grupos de códigos aninhados. Outros dados apresentados na literatura também foram usados.

Para isso, um programa foi escrito na linguagem *PHP* para analisar estruturas internas dos códigos-fontes que respondem os problemas para levantar os seguintes dados:

- Número de estruturas de repetição utilizadas no programa, representado por *N_REP*.
- Número de estruturas de seleção, *N_SEL*.
- Número de linhas de código, *N_LCOD*.
- Número de arestas existentes em um grafo que representa a sequência do algoritmo, *N_ARESTA*.
- Número de arestas que representam estruturas de repetição do grafo/sequência do algoritmo, *N_AREST_REPET*.
- Altura de uma árvore em que cada sub-grupo de códigos aninhados internos refere-se a nó, *ALTU_MAX*.
- Número de assuntos abordados no problema, *N_ASSUNT*. O cadastro dos assuntos foi feito manualmente.

O exemplo do problema *Soma* é descrito no Apêndice D. Seu código-fonte consta na Figura 3.4. As Figuras 3.5 e 3.6 explicam a formação do grafo e da árvore para o código-fonte do problema Soma.

```
1  #include<stdio.h>
2  int main(){
3      int a,b, x;
4      x=0;
5      while(1){
6          scanf("%d %d",&a,&b);
7          if (a==0 && b==0){
8            break;
9          }
10         x = a+b;
11         printf("%d\n", x);
12     }
13     return 0;
14 }
```

Figura 3.4: *Código-fonte do problema Soma*

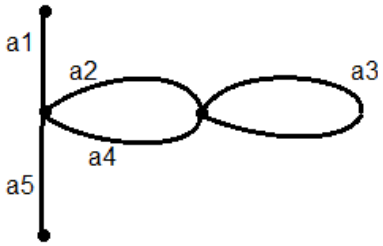

Figura 3.5: *Grafo para o código-fonte do problema Soma*

Figura 3.6: *Árvore para o código-fonte do problema Soma*

O grafo, Figura 3.5, possui 5 arestas, que representam a sequencia do programa relacionado ao problema Soma (Figura 3.4). Cada aresta representa uma parte da sequência do algoritmo. Como a aresta *a1* representa o início do programa, linha dois até quatro, a aresta *a2* representa o início da estrutura de repetição *while*, linha cinco até seis, a aresta *a3* representa a estrutura de seleção *if*, linha 7 até linha 9, a aresta *a4* representa a continuidade da aresta *a2*, linha 10 até 12, e a aresta *a5* representa o fim do programa, linha treze até quatorze.

A árvore, Figura 3.6, pode ser entendida fazendo uma analogia a conjuntos. Considerando a existência de conjuntos que representam blocos de código que possuem

sub-conjuntos internos que representam os blocos aninhados. A árvore representa este aninhamento entre os conjuntos. No exemplo existem três blocos de código: *n1* iniciado na linha dois, *n2* na linha cinco, e *n3* na linha sete. Essa árvore apresenta o formato de uma lista pelo fato de que cada nó possui somente um nó filho, i.e. cada bloco de código possui no máximo um sub-bloco de código aninhado do mesmo nível hierárquico.

O Algoritmo 3.1 foi feito para contar o número de arestas do grafo que representa o programa e o número destas arestas relativas a estrutura de repetição. Ele foi projetado a partir da contagem das arestas no desenho manual do grafo a partir do código-fonte.

Algoritmo 3.1: Contar Arestas do Grafo do Programa

Entrada: arquivoCodigoFonte
Saída: N_ARESTA, N_AREST_REPET

```
1  pilha ← NULL.
2  for linha ∈ arquivoCodigoFonte do
3      if (possuiMain(linha)) then
4          N_ARESTA++.
5      end
6      if (possuiSelecao(linha)) then
7          N_ARESTA ← N_ARESTA + 2.
8          if (topoPilha(pilha) ==' repeticao') then
9              N_AREST_REPET++.
10         end
11         empurre(pilha,' selecao').
12     end
13     if (possuiRepeticao(linha)) then
14         N_ARESTA ← N_ARESTA + 2.
15         N_AREST_REPET++.
16         if (topoPilha(pilha) ==' repeticao') then
17             N_AREST_REPET++.
18         end
19         empurre(pilha,' repeticao').
20     end
21     if (possuiSelecaoElse(linha)) then
22         N_ARESTA++.
23         empurre(pilha,' selecao').
24     end
25     if (possuiFechamentoDeBloco(linha)) then
26         removerTopo(pilha).
27     end
28 end
```

Um arquivo de código-fonte é dado como entrada para o programa. Ele é divido em um conjunto de linhas. Ocorrem iterações nesse conjunto e as arestas são contabilizadas. Há o uso de uma pilha para controlar, em momentos específicos da iteração, sob qual estrutura, como seleção ou repetição, o atual bloco de código se encontra e possibilitar a contagem das arestas relativas à tal.

A função *possuiMain()* verifica se a linha da atual iteração é a declaração do programa. A função *possuiSelecao()* verifica se a linha refere-se à estrutura de seleção. A função *possuiRepeticao()* verifica se trata-se de estrutura de repetição. A função *possuiSelecaoElse()* verifica se trata-se do comando *senão* da estrutura de seleção. E a função *possuiFechamentoDeBloco()* verifica se trata-se do fechamento de bloco de código-fonte com "}".

O Algoritmo 3.2 mede a altura da árvore. Ocorre a consideração de que cada subgrupo aninhado de código-fonte representa um nó filho da árvore. Durante as iterações ocorrem as verificações das existências de início de bloco de código-fonte a partir de "{" e fechamento de bloco de código-fonte com "}", para isso há o uso das funções *possuiInicioDeBloco()* e *possuiFechamentoDeBloco()*. A altura da árvore é atualizada para considerar a relação de dependência.

Algoritmo 3.2: Medir Altura da Árvore que Representa o Aninhamento do Código-Fonte

Entrada: arquivoCodigoFonte
Saída: ALTU_MAX

```
1  alturaTemp ← 0.
2  ALTU_MAX ← 0.
3  for linha ∈ arquivoCodigoFonte do
4      if (possuiInicioDeBloco(linha)) then
5          alturaTemp++.
6          if (ALTU_MAX < alturaTemp) then
7              ALTU_MAX ← alturaTemp.
8          end
9      end
10     if (possuiFechamentoDeBloco(linha)) then
11         alturaTemp−−.
12     end
13 end
```

O exemplo do problema *Soma*, Apêndice D, possui as seguintes propriedades e valores. *N_REP*=1; *N_SEL*=1; *N_LCOD*=14; *N_ASSUNT*=6; *N_ARESTA*=5; *ALTU_MAX*=3; *N_AREST_REPET*=2. Os assuntos que esse problema aborda são: entrada de dados, saída de dados, estruturas de seleção, estruturas de repetição, domínio de lógica de programação e matemática.

Após a extração de dados dos programas, ocorreu a busca de dados sobre as dificuldades definidas pelos alunos em sua interação com o sistema *BOCA*.

Foram selecionados dados que mostrassem as tentativas dos alunos em responder os problemas. Os dados obtidos possuem a seguinte estrutura: tentativa, problema e dificuldade do problema especificada pelo aluno. Esta seleção contou a regra a seguir.

Regra: selecionar somente tentativas de alunos que responderam acima de 70 problemas.

A regra foi incluída devido à existência de alunos que fazem a metade da disciplina e desistem. Isso poderia fazer com que um conjunto de problemas, por não ter respostas de um grupo de alunos, não influenciasse na dificuldade e os dados ficassem desproporcionais. A obtenção dos dados, realizada considerando a regra, foi feita selecionando os dados no BD *PostgreSql* e calculando a média, mediana e moda para cada problema.

De um total de 92 usuários, 62 se enquadraram na regra, o que representa 67.39%. As interações desses usuários com os problemas foram utilizadas.

3.2.3 Resultados

A média das dificuldades dos problemas do ponto de vista dos alunos variou de 1.44 até 4.22 e o desvio padrão foi de 0.566199435.

Os dados obtidos possibilitaram identificar algumas correlações que contribuíram para entender melhor os fatores que influenciam na dificuldade de problemas de CS1 no contexto dos alunos.

As variáveis apresentadas na Tabela 3.4 foram levantadas para todos os 74 problemas. Houve o uso do *software PSPP* para calcular as correlações.

Tabela 3.4: *Correlações entre as variáveis e a dificuldade no contexto dos alunos.*

		MEDIANA_DIFIC	MEDIA_DIFIC	MODA_DIFIC
N_REP	Pearson Correlation	,49	,52	,44
	Sig. (2-tailed)	,000	,000	,000
	N	74	74	74
N_SEL	Pearson Correlation	,16	,34	,19
	Sig. (2-tailed)	,171	,003	,108
	N	74	74	74
N_LCOD	Pearson Correlation	,44	,62	,45
	Sig. (2-tailed)	,000	,000	,000
	N	74	74	74
N_ASSUNT	Pearson Correlation	,57	,69	,59
	Sig. (2-tailed)	,000	,000	,000
	N	74	74	74
N_ARESTA	Pearson Correlation	,42	,58	,41
	Sig. (2-tailed)	,000	,000	,000
	N	74	74	74
ALTU_MAX	Pearson Correlation	,52	,67	,56
	Sig. (2-tailed)	,000	,000	,000
	N	74	74	74
N_AREST_REPET	Pearson Correlation	,53	,60	,51
	Sig. (2-tailed)	,000	,000	,000
	N	74	74	74

Os dados da Tabela 3.4 possuem correlações de dados dos problemas com a média, a moda e a mediana das dificuldades na visão dos alunos. As correlações com a média apresentam resultados melhores do que com a moda e mediana, e por isso decidiu-se usá-la.

É possível observar que *N_ASSUNT*, com 0.69, e *ALTU_MAX*, com 0.67, apresentaram as maiores correlações com *MEDIA_DIFIC*. A variável *N_SEL* apresentou a menor correlação e as demais (*N_LCOD, N_AREST_REPET, N_ARESTA, N_REP*) ficaram no meio. As correlações entre todas as características com *MEDIA_DIFIC* tiveram significância aceitável, o que indica que as correlações são confiáveis.

Algumas equações envolvendo as variáveis foram criadas manualmente para verificar a correlação com as médias de dificuldades dos problemas, *MEDIA_DIFIC*, e são descritas como (*f1, f2, f3, f4*). A análise das correlações das variáveis individuais motivou criar essas equações com o objetivo de encontrar correlações mais significativas. A Tabela 3.5 apresenta as correlações das equações testadas. A equação *f4* teve a maior correlação, 0.76, e significância aceitável.

Tabela 3.5: *Correlações entre as funções.*

		media_dificul
f1	Pearson Correlation	,66
	Sig. (2-tailed)	,000
	N	74
f2	Pearson Correlation	,68
	Sig. (2-tailed)	,000
	N	74
f3	Pearson Correlation	,67
	Sig. (2-tailed)	,000
	N	74
f4	Pearson Correlation	,76
	Sig. (2-tailed)	,000
	N	74

Seguem as descrições das equações apresentadas na Tabela 3.5.

$$f1 = \frac{N\_REP + N\_SEL + N\_LCOD + N\_ASSUNT + N\_ARESTA + ALTU\_MAX + N\_AREST\_REPET}{7} \tag{3-1}$$

$$f2 = N\_ARESTA \cdot 0.3 + ALTU\_MAX \cdot 1.4 + N\_AREST\_REPET \cdot 1.4 + N\_LCOD \cdot 0.2 \tag{3-2}$$

$$f3 = \frac{N\_ASSUNT + ALTU\_MAX + N\_LCOD}{3} + \frac{N\_AREST\_REPET + N\_ARESTA + N\_REP + N\_SEL}{8} \tag{3-3}$$

$$f4 = \frac{N\_ASSUNT + ALTU\_MAX}{2} \tag{3-4}$$

A equação *f4* foi criada usando as variáveis *N_ASSUNT* e *ALTU_MAX*. Foram identificadas correlações interessante entre essas variáveis e a dificuldade do problema e também entre a função *f4* e a dificuldade.

As mesmas variáveis, conforme a Tabela 3.6, que foram analisadas considerando o ponto de vista dos alunos, foram confrontadas com a visão dos professores. As dificuldades foram definidas por quatro professores, e devido a esse conjunto de dados reduzido decidiu-se trabalhar com a mediana.

Tabela 3.6: *Correlações entre as variáveis e a dificuldade no contexto dos professores*

		dif_mediana
N_REP	Pearson Correlation	,62
	Sig. (2-tailed)	,000
	N	74
N_SEL	Pearson Correlation	,20
	Sig. (2-tailed)	,093
	N	74
N_LCOD	Pearson Correlation	,56
	Sig. (2-tailed)	,000
	N	74
N_ASSUNT	Pearson Correlation	,50
	Sig. (2-tailed)	,000
	N	74
N_ARESTA	Pearson Correlation	,53
	Sig. (2-tailed)	,000
	N	74
ALTU_MAX	Pearson Correlation	,50
	Sig. (2-tailed)	,000
	N	74
N_AREST_REPET	Pearson Correlation	,62
	Sig. (2-tailed)	,000
	N	74

A mediana da dificuldade dos problemas, segundo a visão dos professores, variou de um até quatro e teve como desvio padrão 0.857513561.

Os dados da tabela 3.6 possibilitaram perceber que as variáveis que tiveram maior correlação foram *N_REP* e *N_AREST_REPET*. Ambas são referentes às estruturas de repetição nas duas representações. A mudança de contexto alterou a ordem de importância das variáveis quanto às correlações. Somente a variável *N_SEL* teve significância inaceitável.

É possível perceber nos dados da Tabela 3.6 indícios de que os professores envolvidos nesta pesquisa fizeram a classificação de uma maneira viciada na dificuldade da estrutura de repetição, pois as variáveis com maior correlação são N_REP e N_AREST_REPET. Observa-se, em discordância a esses dados, que além da dificuldade dos assuntos é necessário considerar aspectos da dinâmica dos algoritmos e abstração envolvida no domínio do problema.

As únicas variáveis que tiveram menores correlações com a dificuldade, no contexto dos professores comparado ao dos alunos, foram *N_ASSUNT* e *ALTU_MAX*. Essa informação traz indícios de que existem professores que deixam de enxergar a dificuldade relacionada com a abstração envolvida no domínio do problema.

As equações analisadas no contexto dos alunos também foram analisadas no contexto dos professores. A tabela 3.7 mostra essa análise. Todas tiveram significância aceitável. Porém, a equação *f4* neste contexto teve a pior correlação com a mediana da dificuldade.

Tabela 3.7: *Correlações entre as equações e a dificuldade no contexto dos professores.*

		dif_mediana
f1	*Pearson Correlation*	,59
	Sig. (2-tailed)	,000
	N	74
f2	*Pearson Correlation*	,63
	Sig. (2-tailed)	,000
	N	74
f3	*Pearson Correlation*	,59
	Sig. (2-tailed)	,000
	N	74
f4	*Pearson Correlation*	,55
	Sig. (2-tailed)	,000
	N	74

A Tabela 3.8 mostra a diferença na atribuição das dificuldades pelos alunos e pelos professores. Existe uma correlação de 0.69 e significância aceitável. Pode ser percebido na Tabela 3.8 que o valor do *Sig. (2-tailed)* é 0.000, o que indica a inexistência de chances dos valores ocorrerem pelo acaso e traz confiança na correlação. Isso mostra que os professores conseguem entender parcialmente a dificuldade dos alunos.

Tabela 3.8: *Correlações entre a mediana das dificuldades considerando alunos e professores.*

		dif_professor_mediana
dif_aluno_mediana	*Pearson Correlation*	,69
	Sig. (2-tailed)	,000
	N	74

Os resultados apresentados trazem a importância de docentes usarem ferramentas para auxiliar na tomada de decisão. As pesquisas, nesse sentido, contribuem para a construção dessas ferramentas, que podem tornar o ensino mais justo e serem usadas principalmente por professores inexperientes ou que não possuem um olhar crítico para a disciplina.

3.3 Discussão

Este trabalho apresentou uma estratégia para medir a dificuldade de problemas visando a geração de listas de problemas de CS1 em sistemas de Juiz *Online*.

Os professores, sem o uso desses sistemas, fazem isso manualmente. A proposta visa automatizar essa atividade de classificação de dificuldade. Para isso, foi feita uma RSL buscando as características relacionadas aos problemas que se relacionam com

a dificuldade e foram feitos experimentos visando identificar correlações entre essas características e a dificuldade do problema.

Existe uma infinidade de possibilidades de condução desta pesquisa. Porém, a escolha de quais características trabalhar e qual estratégia de pesquisa usar se deve ao conjunto de dados disponível a partir da experiência do BOCA no ensino de CS1 realizado no âmbito deste trabalho.

A Tabela 3.9 mostra como o estudo realizado abordou as características trazidas pela RSL.

Tabela 3.9: *Comparação da abordagem realizada na pesquisa com a literatura.*

CONJUNTO	CARACTERÍSTICA	EXPLICAÇÃO
Erros	Erros dos alunos	
	Erros nos programas	
	Calculo de Percentuais de acertos	
Conceitos	**Conjunto de assunto**	**Contagem do número de assuntos envolvidos no problema**
	Estrutura de repetição	**Contagem do número de estruturas de repetição**
	Estrutura de seleção e Instruções switch	**Contagem do número de estruturas de seleção**
	Matrizes	
	Entrada e saída em arquivos	
	Strings	
	Sequência	**Contagem do número de linhas do código-fonte do programa**
	Consideração de conceitos ensinados no curso até o momento da invenção da atividade	
Estratégias	Professor analisa a dificuldade	
	Análise da solução escrita pelo autor do problema	**Uso da solução, código-fonte, escrita pelo autor do problema**
	Resolução do problema para estimar a dificuldade	
	Estuda a dificuldade para a disciplina de CS1 de uma maneira geral	
	Estuda a dificuldade de um problema específico	**Consideração das características de cada problema**
Resolução de Problemas	Raciocínio lógico	
	Capacidade de síntese a partir da leitura do código-fonte	
	Dificuldade de lidar com o problema sem o computador	
	Compreensão textual	**Uso de grafo e árvore, que relaciona-se com aspectos semânticos do problema**

Este trabalho não aborda a capacidade cognitiva do aluno no momento em que ele classifica a dificuldade de um problema. Isso é necessário para possibilitar investigar o problema em maior profundidade. Porém, os dados obtidos não possuem tais informações, e a estratégia proposta (equação *f4*) possui resultados satisfatórios para o contexto trabalhado.

CAPÍTULO 4

Plágio em código-fonte de submissões

Lidar com plágio em códigos-fonte de programas é um desafio. Esse desafio começa com a definição de plágio. Copiar parte ou todo um programa e apresentá-lo como seu é considerado plágio, mesmo quando o programa é o resultado de um trabalho conjunto (Wagner [65] e Joy e Luck [35]). Além disso, existem casos de plágio em que se fazem modificações estruturais nos programas, mantendo as mesmas funcionalidades. Porém, citar fontes e dar o devido crédito ao autor pode ser o suficiente para passar de plágio para referência (Bakshi [12]).

Diversos relatos mostram que o plágio em programas de computadores é maior que em outras áreas [13, 65, 12]. Baugh et al. [13] relatam que na *Stanford University* 23% das violações de seu código de honra foram feitas por alunos de CS1, enquanto que estudantes de Ciência da Computação são somente 6.5% do total de alunos. Na *Brown University*, 70% dos casos de plágio analisados vieram do departamento de Ciência da Computação (Bakshi [12]). Wagner [65] apresenta casos semelhantes no *Massachusetts Institute of Technology* (MIT) e na *University of Texas*. No entanto, isso pode ser porquê muitos cursos de Computação checam sistematicamente a cópia de programas de maneira automática, usando *software* específico, o que não ocorre em outras áreas (Bakshi [12]).

Vale ressaltar que o plágio em programas tem algumas características que o diferenciam do plágio em outras disciplinas. Os programas, muitas vezes são bastante simples e são implementados em ambientes similares. Problemas mais simples implicam em códigos-fonte menores e mais parecidos. Por usarem uma linguagem de programação, que possui um conjunto de comandos limitado e sintaxe bem definida, a liberdade de mudança não é a mesma que em um texto. Além disso, em muitos casos, o enunciado do problema tem formatos da entrada e de saída pré-definidos limitando ainda mais a possibilidade de diferenciação do código. Este cenário, frequentemente encontrado na disciplina de Introdução à Programação de Computadores, traz dificuldades na detecção de plágio.

Várias razões são dadas para "justificar" o plágio [65] e estão ligadas a aspectos diversos como: organização da disciplina, currículo do curso, professores, ambiente de programação, e problemas pessoais. Razões incluem: *deadline* apertado, com pouca

ou nenhuma nota ofertada para soluções parciais; alunos consideram o trabalho chato ou muito difícil; reprovação quando os exercícios não são entregues, independente da nota dos exames; uso de listas de exercícios de turmas anteriores; ter a visão de que programação é irrelevante em suas carreiras; falta de base; entre outros.

Os resultados apresentados na pesquisa realizada com 115 estudantes da área de informática em uma universidade privada dos Estados Unidos [13] mostraram que: 52% dos alunos acreditam que copiar código-fonte da *internet* é errado, mas apenas 30% dos alunos acreditam que é preciso citar os códigos baixados da *internet*; 78% dos alunos disseram já ter pesquisado código-fonte na *internet* como apoio à atividades, e 17% dos alunos confirmaram já terem usado os códigos-fonte baixados como respostas em atividades do curso. Os estudantes pesquisados consideram ser mais errado copiar dos colegas do que pesquisar as respostas na *internet*.

Uma pesquisa feita na *School of Computer Science and Software Engineering* da *Monash University* e na *School of Information Technology* da *Swinburne University* na Austrália [58] aplicou um questionário com 18 situações e pediu aos alunos para classificarem, usando uma escala *Likert* de 5 pontos, se eles achavam que aquela situação era aceitável do ponto de vista ético. Na grande maioria das situações, envolvendo cola em exames, roubo e plágio explícito, os alunos concordaram que as situações não eram aceitáveis. No entanto, em algumas situações as opiniões variaram. Casos onde dois alunos colaboravam para resolver um problema que deveria ser resolvido individualmente, pedir ajuda através da *internet*, e submeter trabalhos de alunos de outros cursos ou turmas, foram considerados aceitáveis por vários alunos.

Mesmo junto a professores há diferentes opiniões sobre o aspecto de colaboração na realização de programas. *Georgia Tech* permite que seus alunos façam trabalhos colaborativos, mas depois pede que os alunos façam demonstrações orais do funcionamento do *software* (Baugh et. al. [13]). O mesmo ocorre na *Monash University* [58]. Em outras universidades, onde a avaliação é feita através de submissão eletrônica, a detecção de plágio costuma ser feita de maneira automática, como é o caso da *University of Swinburne*. Nessas situações, cópia de programas não é aceitável.

Um estudo apresenta uma lista de técnicas usadas por alunos para plagiar (Joy e Luck [35]). Estas incluem (1) mudanças léxicas, que podem ser feitas em um editor de texto e não exigem conhecimento da linguagem, como por exemplo inclusão, modificação ou remoção de comentários, mudanças na formatação, e troca de nomes de variáveis, ou (2) mudanças estruturais, que exigem maior conhecimento, incluindo a capacidade de analisar o código (*parse*), como por exemplo, mudar o tipo de *loop* (*for/while*), substituir *if's* aninhados por comandos do tipo *case*, trocar a ordem dos comandos sem afetar a estrutura, substituir chamadas a procedimentos pelo próprio procedimento, trocar a ordem dos operadores (e.g. x<y pode ser substituído por y >= x).

Outro trabalho faz análise semelhante. No artigo, os autores afirmam que é possível analisar a similaridade entre códigos-fonte a partir da semelhança representacional ou comportamental (Mota e Goya [45]). A semelhança representacional é sintática e refere-se ao programa sendo uma sequência de caracteres, enquanto a semelhança comportamental é semântica e pode ser definida pelas funções que os programas implementam. As ferramentas para análise de similaridade diferenciam-se no método de comparação e qual tipo de semelhança se pretende analisar. Esse cenário, aliado à dificuldade na definição dos tipos de similaridade, traz desafios para a pesquisa sobre plágio e a necessidade de análise humana durante o processo.

No contexto específico de disciplinas de CS1, a maioria dos casos de plágio utilizam mudanças léxicas, que são mais fáceis de serem realizadas a partir de alterações nos arquivos dos programas em editores de texto ou ambientes de desenvolvimento. Mudanças estruturais exigem um conhecimento de programação que os alunos muitas vezes ainda não possuem, e que na maioria das vezes é a razão que leva o aluno a plagiar.

Este capítulo apresenta uma estratégia para apoio à identificação de plágio em códigos-fonte de atividades de CS1. O contexto de ensino envolvido nesta pequisa trabalha inicialmente com problemas simples, que possuem soluções com poucas linhas de código, e que vão aumentando gradativamente de nível de dificuldade. Essa simplicidade traz requisitos adicionais que devem ser tratados e serão discutidos no decorrer do trabalho.

A estratégia proposta, que visa ser parte de um sistema de apoio à identificação de plágio na disciplina de CS1, contribui para resolver o problema em questão e é fortemente adaptada ao seu contexto. Isto é, ela foi desenvolvida para trabalhar com programas simples escritos em C, com poucos comandos; foca o plágio do tipo léxico, mais comum em disciplinas introdutórias e verificadas na prática do dia-a-dia na sala de aula; é de fácil uso, sem exigência de configurações complicadas; é *open-source*; e é executada localmente, podendo ser incorporada no sistema.

É uma proposta simples e eficaz, com ênfase no pré-processamento dos programas submetidos. A solução adota um algoritmo conhecido, algoritmo de Distância de Edição, e um conjunto restrito de normalizações, o que facilita sua implementação e reprodução. O pré-processamento foi definido a partir da análise de normalizações propostas na literatura e refinado através de vários testes usando o banco de problemas disponível.

4.1 Revisão Sistemática da Literatura

A RSL descrita nesta seção tem a finalidade de apresentar o estado da arte sobre plágio e a disciplina de CS1. Para isto, houve a intenção de responder às seguintes questões de pesquisa.

- **QP:** Qual é a estratégia mais adequada para identificar plágio em códigos-fonte de atividades de CS1?

- **QP1:** Quais estratégias já foram usadas para identificar plágio em códigos-fonte de atividades de CS1?

- **QP2:** Como validar a estratégia usada para identificar plágio em códigos-fonte de atividades de CS1?

A RSL seguiu as instruções elaboradas por Kitchenham [36] e analisou trabalhos publicados no período de 2010 a 2016. A busca foi realizada na data 28/02/2016. A estruturação dos passos da RSL foi apresentada por Vieira [63] na Figura 2.1.

4.1.1 Planejamento

O protocolo consta no Apêndice B.

4.1.2 Execução

Foram identificados 30 trabalhos. A Figura 4.1 mostra a distribuição dos trabalhos de acordo com as bases de pesquisa e a Figura 4.2 mostra a distribuição por base de dados e ano.

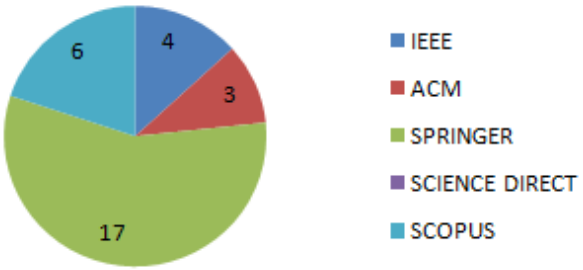

Figura 4.1: *Número de Trabalhos identificados agrupados por base de pesquisa.*

As bases de pesquisas *Springer* e *Scopus* novamente foram as que mais trouxeram resultados.

A Figura 4.2 mostra que a partir de 2015 houve um aumento nas publicações sobre o assunto.

A seleção contou com a leitura do título, palavras-chave e resumo de cada trabalho e uso dos critérios de inclusão e exclusão para definir os trabalhos usados na análise completa. A Figura 4.3 mostra os 25 trabalhos excluídos conforme critérios de exclusão. Os critérios foram: *E1 - Texto completo não disponível para acesso gratuitamente na web ou no portal de periódicos da Capes*; *E3 - Trabalho não aborda plágio e CS1*; *E5 - Publicação duplicada*. Sobraram 5 artigos após execução dessa etapa.

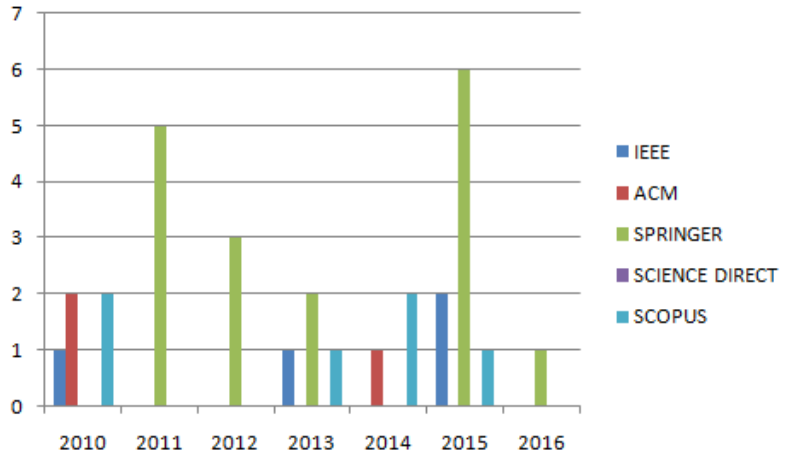

Figura 4.2: *Número de Trabalhos identificados agrupados por ano e base de pesquisa.*

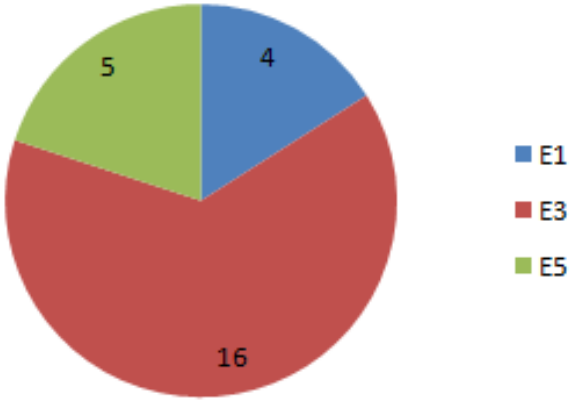

Figura 4.3: *Trabalhos removidos na seleção conforme critérios de exclusão.*

A extração identificou 4 trabalhos para responder às questões de pesquisa. Nesta fase foi excluído um trabalho com o critério *E3 - Trabalho não aborda plágio e CS1*. A Tabela 4.1 mostra como os trabalhos restantes se distribuem nos critérios de inclusão (*I1 - Aborda estratégias usadas para identificar plágio em códigos-fonte de atividades de CS1*; *I2 - Aborda validação de estratégias usadas para identificar plágio em códigos-fonte de atividades de CS1*).

4.1.3 Conclusão

Avaliar os resultados de cada trabalho considerando os critérios da RSL permitiu sintetizar os dados. Com isso, foi formada uma visão geral sobre o contexto das pesquisas.

QP1: Quais estratégias já foram usadas para identificar plágio em códigos-fonte de atividades de CS1?

Tabela 4.1: *Trabalhos incluídos na extração conforme critérios de inclusão.*

CRITÉRIO	REFERÊNCIAS	Nº TRABALHOS
I1	[7] [28] [51] [46]	4
I2	[7] [28] [46]	3

A Tabela 4.2 apresenta as características usadas em estratégias de detecção de plágio identificadas a partir da leitura completa dos trabalhos.

Tabela 4.2: *Características encontradas nas estratégias para identificar plágio.*

CARACTERÍSTICA	REFERÊNCIAS	Nº TRABALHOS
Uso do Programa MOSS	[51]	1
Pré-Processamento	[46, 28]	2
Attribute Counting Metrics	[7]	1
Uso de Frequência de Termo	[46, 28]	2
Uso de Distâncias	[46, 28]	2
Equivalence Ratio Calculation	[7]	1

A ferramenta *MOSS* foi usada para identificar plágio em um sistema de avaliação automática de programas usado por alunos para fazerem tarefas para casa (Pozenel et. al. [51]).

Foi possível perceber uma preocupação com o pré-processamento. Ohmann e Rahal [46], no projeto do sistema *PIY*, trabalham com a remoção de comentários e espaços em branco em excesso nos códigos-fonte e a criação de uma sequência de átomos de tamanho k, gerando os k-gramas. Já Gaudencio et. al. [28] trabalharam com a extração de átomos da árvore de sintaxe abstrata de um código-fonte *Python*.

Al-Khanjari et. al. [7] projetaram uma estratégia de processamento baseada em *Attribute Counting Metrics* (ACM) para a ferramenta *PlagDetect*. Técnicas ACM comparam ocorrências de algumas características do código-fonte do programa. Nesse trabalho, há um experimento que usou palavras-chave únicas, métodos únicos, interfaces únicas, construtores únicos, número total de palavras-chave, número total de métodos, número total de interfaces, número total de construtores, número total de blocos de instrução e número total de instruções de atribuição.

O uso de Frequência de Termo foi observado nos trabalhos [46, 28]. O sistema *PIY* [46] trabalha com comparações com base em listas de frequência dos átomos de tamanho k, os k-gramas. As listas de frequência dos k-gramas são tratadas como vetores com posições que representam k-gramas e valores que representam a frequência. Essa técnica também é usada na área de mineração de texto e recuperação de informações. Gaudencio et. al. [28], em uma pesquisa cuja finalidade é responder à pergunta *computadores conseguem comparar soluções de código-fonte de estudantes como professores?*, usaram

quatro variações de frequência do termo (contagem normal de termos, contagem aumentada de frequência de termo que leva em conta cada frequência de acordo com a maior frequência em um documento, contagem binária de termos que definem a frequência de um termo, média *log* de frequências de termo)

Houve o cálculo de distâncias nos trabalhos [46, 28]. Gaudencio et. al. [28] trabalharam com distância de *Jaccard*, distância de edição de texto e distância de edição de árvore. Já Ohmann e Rahal [46] trabalharam com as distâncias Euclidiana, *Manhattan* e *co-seno*. Essas distâncias são métricas que calculam a diferença entre dois elementos.

Al-Khanjari et. al. [7] projetaram uma estratégia de cálculo da similaridade baseada em um esquema estatístico feito a partir de *Equivalence Ratio Calculation* e que usou ACMs.

QP2: Como validar a estratégia usada para identificar plágio em códigos-fonte de atividades de CS1?

A Tabela 4.3 resume as características encontradas na validação das ferramentas e quantifica os trabalhos que as abordam. As características são contextualizadas no texto que segue.

Tabela 4.3: *Características encontradas nas validações das estratégias para identificar plágio.*

CARACTERÍSTICA	REFERÊNCIAS	Nº TRABALHOS
Ameaças às validações	[28, 7]	2
Conjuntos de códigos-fonte agrupados por características semelhantes	[7]	1
Escolha do valor mais apropriado para tamanho do átomo	[46]	1
Falsos-positivos e falsos-negativos	[7, 46]	2
Precision, *Recall* e *F1 score*	[46]	1
Comparação com outra ferramenta	[7, 46]	2

As pesquisas [28, 7] trouxeram ameças que devem ser consideradas nas validações das ferramentas. Gaudencio et. al. [28] apontam as ameaças ao concluir que o acordo entre os professores, ao classificarem a similaridade entre programas de CS1, não é alto e que cada professor faz classificações semelhantes à certas estratégias automáticas. O acordo entre professores variou de 62% a 95% enquanto o acordo entre os algoritmos e professores é em todos os casos superior à 75%. Já o estudo de Al-Khanjari et. al. [7] mostra a necessidade de estudar a linha tênue entre os níveis aceitável e inaceitável de semelhanças entre os programas. Acredita-se que a definição desta linha para a disciplina de CS1 depende de vários motivos relacionados ao ambiente de programação. Percebe-se uma dependência do contexto de ensino e a influência de fatores subjetivos na validação.

Diversas ações podem ser realizadas para lidar com as ameaças. A validação realizada para a ferramenta *PlagDetect* [7] trabalhou com conjuntos de programas agrupados

por características semelhantes, como um conjunto de programas pequenos. Já Ohmann e Rahal[46] fizeram um estudo experimental que visa escolher o valor mais apropriado para o tamanho do átomo que será usado no processo de comparação, que tem sua variação dependente do contexto de ensino.

Houve o uso de dados de falsos-positivos e falsos-negativos na validação da ferramenta *PlagDetect* [7]. A validação do sistema *PIY* [46] foi feita a partir de *precision*, *recall* e *F1 score*, variáveis que se relacionam com falsos-positivos e falsos-negativos.

Houve comparações de resultados de ferramentas. A validação da ferramenta *PlagDetect* [7] comparou seus resultados com os da ferramenta com *JPlag* e a validação do sistema *PIY* [46] comparou seus resultados com os do sistema *MOSS* e mostrou que o sistema *PIY* possui resultados melhores para os experimentos realizados.

QP: Qual é a estratégia mais adequada para identificar plágio em códigos-fonte de atividades de CS1?

As estratégias para identificar plágio em códigos-fonte de atividades de CS1 podem ter várias configurações. A realização de pré-processamento foi verificada nos trabalhos de Ohmann e Rahal [46] e Gaudencio et. al. [28]. Para fazer as comparações, há o processamento da análise de similaridade no texto como um todo [46] e, por outro lado, o em características específicas do texto denominadas "ACMs" (Al-Khanjari et. al.[7]).

Al-Khanjari et. al. [7] acreditam que a definição da linha tênue entre os níveis aceitável e inaceitável de semelhanças entre os programas depende de vários motivos relacionados ao ambiente de programação. Isso mostra que para responder à pergunta *Qual é a estratégia mais adequada para identificar plágio em códigos-fonte de atividades de CS1?* é preciso explorar as diversas possibilidades de estratégias nos contextos de ensino possíveis, que envolvem diferentes linguagens de programação e ferramentas.

Existe uma alta discordância entre professores ao classificarem a similaridade dos programas (Gaudencio et. al. [28]). Isso impacta diretamente nas comparações de resultados entre ferramentas, como fizeram [7, 46]. Validar tais ferramenta depende de um planejamento que tenha essas preocupações e da análise dos resultados por meio de variáveis como falsos-positivos, falsos-negativos, *precision*, *recall* e *F1 score*.

4.2 Estudo sobre Ferramentas

Existem vários programas para apoiar a identificação de plágio em código-fonte. Entre eles: *CodeMatch, CPD, JPlag, Marble, MOSS, Plaggie, Sherlock, SIM, SID* e *YAP3* [2, 6, 18, 21, 33, 52, 57, 69, 70]. Apesar de implementarem algoritmos distintos, algumas estratégias e conceitos são comuns.

Os programas calculam um índice de similaridade entre textos de códigos-fonte e, com isso, oferecem suporte para a verificação de plágio. Os programas possuem as eta-

pas de pré-processamento, processamento e pós-processamento (Kleiman [37]). O pré-processamento visa gerar uma sequência de átomos para o processamento, e eventualmente trabalha com regras de normalização. O processamento compara os conjuntos de átomos das duas submissões usando um algoritmo. E o pós-processamento calcula um grau de similaridade a partir dos resultados obtidos nas comparações, apresentando-o para o usuário.

O conceito de átomo, do inglês *token*, é frequentemente usado nas estratégias de análise de similaridade. Kleiman [37] o define como elementos indivisíveis de uma cadeia ou sequência. O que é considerado um átomo pode variar. *Hash* é outro conceito bastante usado nos algoritmos de análise de similaridade. Trata-se de uma função que recebe como entrada um conjunto de dados, e.g. um arquivo ou uma *string*, e o transforma em uma sequência menor de informações, que pode ser representada por uma *string* ou uma sequência de bits. Esse conceito é bastante usado na área de Criptografia e Segurança da Informação.

CodeMatch é um programa que usa força bruta para calcular a correlação entre pares de código-fonte usando diversos aspectos do código. Ele combina cinco algoritmos de comparação [70, 56]: *Statement Matching*, *Comment/String Matching*, *Instruction Sequence Matching*, *Identifier Matching*, *Correlation Score*. As similaridades entre os pares são apresentadas como uma pontuação que varia de 0 a 100.

O algoritmo *Statement Matching* compara cada linha funcional do código-fonte de ambos os arquivos, excluindo as linhas de comentário e as linhas que possuem somente palavras reservadas da linguagem. O algoritmo conta o número de linhas de instrução correspondentes nos dois arquivos. O algoritmo *Comment/String Matching* compara cada linha de comentário e cada sequência de caracteres de ambos os arquivos. As sequências de espaços em branco são convertidos em espaços individuais. O algoritmo *Instruction Sequence Matching* compara a primeira instrução de cada linha de origem no par de arquivos, ignorando linhas em branco e linhas de comentários. A função do *Identifier Matching* é comparar o número de identificadores que não são palavras reservadas da linguagem de programação. Por fim, *Correlation Score* determina uma correlação para a semelhança entre os pares de arquivos.

O programa *CPD* [21] foi desenvolvido para identificar partes duplicadas em um mesmo código-fonte, criadas a partir de copiar/colar, quando a mesma funcionalidade é reutilizada. Essas partes copiadas trazem problemas para a manutenção de *software*, além de dificultar o reuso. Apesar de possuir alguns aspectos relacionados com a identificação de plágio, é importante ressaltar que foi projetado para atender a requisitos diferentes.

Para analisar a similaridade entre as partes, no *CPD*, é criada uma tabela que possui uma coluna com sequências de átomos agrupados em uma *string* e outra coluna contendo o conjunto de posições em que a *string* aparece no código-fonte. A comparação

para identificar a similaridade é feita com base nessas tabelas.

O *JPlag* [52] compara um fluxo contínuo de átomos de um arquivo com partes extraídas do outro. O valor da similaridade é calculado a partir das semelhanças encontradas. No pré-processamento do *JPlag*, os arquivos de código fonte são normalizados. Para isso ocorrem as remoções de espaços em branco, comentários e nomes de identificadores. Após essa normalização, faz-se a adição de informação semântica, que expressa a finalidade dos comandos, nos átomos quando possível.

O método utilizado no processamento do *JPlag* é o *String Tiling Greedy*. Seu objetivo é encontrar os maiores conjuntos de sequências de *strings* que se repetem nos dois arquivos e não se sobrepõem. Devido à complexidade computacional de encontrar essas ocorrências maximais de *substrings*, *String Tiling Greedy* é um algoritmo heurístico, i.e, visa trabalhar com os melhores resultados possíveis, que nem sempre são ótimos.

Plaggie é um programa criado para identificar a possibilidade de plágio entre códigos-fonte escritos em Java Ahtiainen et. al. [6]. A diferença do *Plaggie* com o *JPlag* é que o *Plaggie* é *open-source* e precisa ser instalado localmente.

O *YAP3* trabalha com as impressões digitais dos arquivos de código-fonte na análise de similaridade. Para isso utiliza a metodologia *Running-Karp-Rabin Greedy-String-Tiling* (RKS-GST) Wise. [69]. O método empregado no *YAP3* também realiza uma normalização, no pré-processamento. A normalização remove dos códigos-fonte os comentários e *strings* constantes, transforma as letras maiúsculas em minúsculas, modifica os sinônimos para uma forma comum, reordena as funções para a sua ordem de chamada, e retira todos os átomos que não são da linguagem específica em análise. A estratégia para calcular a semelhança entre os arquivos busca um conjunto de regiões similares nos arquivos que não se sobrepõem. Nessa fase, ele usa o algoritmo guloso *RKS-GST*, que trabalha com impressões digitais em *hash-table*. A estratégia de comparação possui custo O(n^2).

É importante observar que os programas *JPlag, Plaggie* e *YAP3* utilizam a estratégia *Greedy-String-Tiling* no seu método de comparação.

O programa *Marble* [33] foi desenvolvido em 2002 pela *Utrecht University* com finalidade de ser simples e de fácil manutenção para detectar casos suspeitos de similaridade em submissões de atividades de alunos. Ele faz normalizações nos arquivos gerando dois arquivos para cada código-fonte e computa a similaridade a partir do comando *diff* do Unix/Linux. O comando *diff* apresenta as linhas de um arquivo de texto que são diferentes das do outro. A similaridade é calculada a partir da razão entre esses números de linhas diferentes e o número total de linhas.

Marble faz suas normalizações analisando a estrutura do programa. Comentários, espaços em branco excessivos, *strings* constantes e declarações de importação são removidos. Outros símbolos são abstraídos ao seu "tipo", por exemplo, cada número he-

xadecimal é substituído por H e cada literal por L. O sistema computa duas versões normalizadas para cada arquivo. Na primeira versão, a ordem dos campos, métodos e classes internas é preservada como no arquivo original. Já na segunda, existem grupos para cada um desses componentes e esses são ordenados de maneira heurística. O usuário precisa escolher se usará somente uma versão normalizada ou se quer trabalhar com as duas. Essa escolha influencia nos resultados, pois o aluno pode ou não mudar a ordem de estruturas internas ao fazer plágio.

MOSS foi desenvolvido em 1994 em *Stanford University* e trabalha com comparação das impressões digitais dos documentos (Schleimer et. al. [57]). O funcionamento do *MOSS* pode ser entendido pelos seguintes passos: (a) é feita uma divisão do documento em *substrings* contíguos de tamanho *k*, sendo *k* escolhido pelo usuário; (b) aplica-se a função *hash* para cada *substring* gerada, formando as impressões digitais do documento; (c) seleciona-se as *hashs* conforme densidade configurada pelo usuário. Para isso é preciso calcular *(hash mod valor_configurado)*. Quanto menor o valor configurado, maior será o conjunto de *hashs* para comparação; (d) tendo executado os passos anteriores para os dois arquivos, calcula-se a similaridade com base no número de *hashs* semelhantes nos arquivos, considerando a densidade escolhida.

A *hash* gerada no *MOSS* é um valor numérico gerado a partir de um algoritmo que recebe uma *string*. Comparar *hashs* exige menos custo computacional do que comparar *strings* caractere a caractere. O conceito de densidade tem a finalidade de reduzir a quantidade de *hashs* a serem comparadas e é implementado a partir da verificação de se a *hash* é divisível pelo valor configurado. Para isso é usado o resto da divisão (*mod*). É preciso configurar os parâmetros de densidade e tamanho das *substrings*. Essa configuração impacta no tempo de processamento e na qualidade dos resultados, e depende do material a ser analisado.

O programa *Sherlock* [2] foi proposto para auxiliar na detecção de plágio de textos a partir de similaridade. Ele trabalha com assinaturas digitais em partes do texto (Maciel [41]). Na fase de pré-processamento, é feita a separação das palavras do texto em grupos de tamanho configurado pelo usuário. Cada conjunto de palavras é convertido em uma assinatura digital por meio de uma função *hash*. Na etapa de processamento, comparam-se as assinaturas digitais dos dois arquivos com a finalidade de calcular a similaridade.

Existem no *Sherlock* parâmetros a serem configurados pelo usuário. *Zerobits* refere-se à granularidade da comparação. Diminuir esse número implica em ter uma comparação mais exata e mais lenta. *Número de palavras* refere-se à quantidade de palavras que se deseja agrupar.

Maciel [41] propõe o sistema *Sherlock N-Overlap*, que modifica o programa *Sherlock* para uso no contexto de plágio em respostas de problemas de programação

introdutória. Para isso, foi feita uma modificação do coeficiente de similaridade, e a inclusão de quatro técnicas de normalização. As técnicas de normalização removem gradativamente partes específicas do código-fonte com a finalidade de preservar estruturas internas do mesmo. A pesquisa conclui que essa modificação estratégica possui resultados melhores comparados com o método tradicional do programa *Sherlock* quando aplicado a programas.

SIM [33] é um programa criado pela *VU University Amsterdam* para detectar plágio em códigos-fonte. É fortemente adaptado à situação de sua Universidade e por isso não é muito portátil. Ele divide cada código-fonte em um conjunto de átomos, formando tabelas de átomos para comparação das correspondências similares. Sua interface com usuário é por linha de comando, e possui diversos parâmetros, os quais, para serem configurados, exigem que o usuário tenha um certo conhecimento de sua estrutura interna (Grune [32]).

O programa *SID*, criado na *University Waterloo*, aceita submissões em Java e C++ (Chen et. al. [18]). Na fase de pré-processamento, faz uso de um analisador léxico específico da linguagem com finalidade de gerar sequências de átomos para o processamento. O processamento é feito com base na medida de informação compartilhada. Essa estratégia foi proposta a partir do conceito de complexidade de *Kolmogorov*. Sua estratégia de processamento utiliza um processo que foi inicialmente implementado para análise de DNA. Esse processo faz uso do algoritmo de Distância de Edição.

Um resumo das estratégias de pré-processamento para a normalização dos arquivos é apresentado na Tabela 4.4.

Tabela 4.4: *Estratégias de normalização.*

LEGENDA			
1	Remover comentários	10	Ordenar elementos internos do código-fonte
2	Remover linhas que possuem somente palavras reservadas da linguagem	11	Padronizar espaço entre elementos do código
3	Gerar sequência de token	12	Remover todos os caracteres situados entre aspas
4	Remover espaços em branco	13	Remover todos os valores literais
5	Remover nomes de identificadores	14	Remover palavras reservadas da linguagem
6	Adicionar informação semântica nos tokens	15	Remover palavras que não são da linguagem
7	Remover strings constantes	16	Remover constantes de caracteres
8	Remover declarações de importação	17	Converter letras maiúsculas para minúsculas
9	Substituir símbolos por seu tipo, e.g hexadecimal é substituido por H e literal por L	18	Converter sinônimos para uma forma comum
		19	Reordenar as funções para sua ordem de chamada

É importante observar que a etapa de pré-processamento tem forte influência na qualidade da análise de similaridade. Kleiman [37], em sua conclusão, propõe que essa etapa é até mais importante do que a aplicação do algoritmo em si. Isto é corroborado por

Maciel [41], que concluiu que as regras de normalização criadas melhoraram significativamente os resultados do *Sherlock*.

A Tabela 4.5 apresenta uma comparação entre os programas discutidos. A Tabela mostra se o programa detecta mudanças léxicas, e/ou mudanças estruturais, e um resumo do pré-processamento que ele realiza usando como referência a Tabela 4.4. As mudanças léxicas são feitas quando é gerado um novo programa com código-fonte diferente do anterior e com a mesma estrutura, como quando há mudanças em nomes de variáveis e inclusão de comentários. Já as mudanças estruturais podem ser percebidas quando um trecho de código tem sua estrutura alterada mas a semântica permanece a mesma, como quando uma equação matemática possui sua estrutura alterada e preserva as mesmas propriedades da anterior.

Tabela 4.5: *Comparação de estratégias usadas nos programas para análise de similaridade.*

PROGRAMA	DETECTA MUDANÇAS LÉXICAIS	DETECTA MUDANÇAS ESTRUTURAIS	PRÉ-PROCESSAMENTO
CodeMatch	SIM	NÃO	1, 2
CPD	SIM	NÃO	3
Jplag	SIM	NÃO	1, 4, 5, 6
Plaggie	SIM	NÃO	1, 4, 5, 6
Marble	SIM	SIM	1, 4, 7, 8, 9, 10
MOSS	SIM	NÃO	* Informação Indisponivel
Sherlock	SIM	NÃO	* Informação Indisponivel
Sherlock N-Overlap	SIM	NÃO	1, 4, 5, 8, 11, 12, 13, 14.
SIM	SIM	NÃO	* Informação Indisponivel
YAP3	SIM	SIM	1, 16, 17, 18, 19, 15
SID	SIM	NÃO	3

Marble, como pode ser visto na Tabela 4.5, detecta algumas mudanças estruturais nos programas, e para isso reordena elementos internos das classes como parte do pré-processamento. Já o *YAP3* detecta as mudanças estruturais relacionadas à ordem de funções. Para tratar essa questão ele reordena as funções conforme a ordem de chamada. Porém, esses programas não conseguem detectar alguns tipos de mudanças estruturais, como a troca da ordem dos operadores e a mudança da ordem dos comandos em um programa sem afetar o resultado.

Uma análise das ferramentas para a identificação de plágio, visando sua integração em um sistema que trabalha com programas de computadores foi feita por Martins [43]. As características comparadas foram: (1) Linguagens suportadas; (2) Possibilidade de adicionar linguagens; (3) Qualidade dos resultados; (4) Interface; (5) Possibilidade de

ignorar o código-base; (6) Submeter grupos de arquivos para processar; (7) Possibilidade de executar o programa local, sem depender de acesso à internet; e se a ferramenta é (8) *Open source*. A Tabela 4.6 mostra o resultado desse levantamento.

Tabela 4.6: *Comparação de programas para analisar similaridade [43]*

PROGRAMA	1 Linguagens suportadas	2 Possibilidade de adicionar linguagens	3 Qualidade dos resultados	4 Interface	5 Possibilidade de ignorar o código-base	6 Submeter grupos de arquivos para processar	7 Possibilidade de executar o programa local	8 Open source
CodeMatch	36	-	X	X	X	-	X	-
CPD	6	X	-	X	?	?	X	X
JPlag	6	-	X	X	X	X	-	-
Marble	5	X	X	-	X	?	X	-
MOSS	25	-	X	X	X	X	-	-
Plaggie	1	?	?	X	?	?	X	X
Sherlock	1	-	-	-	-	-	X	?
SIM	7	?	X	-	-	-	X	?
YAP	5	?	X	-	X	?	X	-

Os dados da Tabela 4.6 apresentam 6 programas com bons resultados na qualidade dos resultados (característica 3). Porém nenhum dos programas classificados com bons resultados são *open-source* e podem ser executados localmente, características necessárias para uso com sucesso em outros projetos.

Para atender aos requisitos do projeto relacionado a esta pesquisa é preciso que o programa possa ser baixado e usado sem nenhum impedimento legal, e para isso é necessário que as características (7) e (8) existam. Esse cenário motiva a criação de uma nova ferramenta.

Além disso, os testes apresentados mostraram que conforme são simulados plágios mais bem elaborados, os programas vão se mostrando insuficientes para identificá-los Martins et. al. [43]. O programa *MOSS*, por exemplo, se esforça muito para tratar os falsos-positivos e tem muitos problemas com diferentes tipos de plágio. O *Sherlock* tem problemas com comentários nos códigos-fonte por não ignorá-los. Já o *CodeMatch* tem problemas com mudança de identificadores nos códigos-fonte.

O problema em lidar com a mudança de nomes de identificadores na ferramenta *CodeMatch* poderia ser abordado usando normalização, ao remover os nomes e usos das variáveis nas submissões. Mas isso faria os códigos-fontes de submissões se tornarem mais parecidos, recaindo no problema de gerar muitos falsos positivos, casos em que o programa classifica como plágio situações que não são.

Kleiman [37] identifica, a partir de testes com ferramentas, uma vulnerabilidade no pré-processamento, que a depender da estratégia usada pode ignorar grande parte da estrutura do programa ou deixar de descartar partes sem importância. Foi proposto um pré-processamento que envolve ordenação lexicográfica do programa. Essa ordenação é feita na árvore sintática do programa de maneira recursiva em uma abordagem de baixo para cima seguindo convenções. Após essa ordenação é gerada uma sequência de átomos e a árvore é destruída.

Essa estratégia foi proposta após encontrar problemas na primeira abordagem, que em um primeiro momento traduzia cada átomo do programa em caracteres ASCII a partir da análise léxica e ignorava nomes de identificadores e, em outro momento, houve a preocupação de relacionar cada caractere gerado pela análise léxica com uma posição no programa original.

Entre os algoritmos que foram usados no processamento das ferramentas, percebe-se a frequente referência ao *RKS-GST*.

Com o objetivo de validar as ferramentas e a solução proposta, foi feito um levantamento buscando as bases disponíveis de programas classificados que pudessem ser usadas. Infelizmente, nenhuma foi encontrada, fator este que pode ser encarado como um desafio para pesquisas nesta área.

4.2.1 Experiência com Sherlock NOverlap

O programa *Sherlock NOverlap* é resultado de uma pesquisa de mestrado que abordou o problema de plágio na disciplina de CS1 (Maciel [41]). Ele não foi encontrado para *download*. Foi necessário realizar modificações no programa *Sherlock* e implementar quatro regras de normalização para adequá-lo ao *Sherlock NOverlap*.

Para ter resultado satisfatório, é preciso usar a configuração mais adequada a um determinado problema. São possíveis diferentes configurações para serem usadas no programa. Essas configurações incluem: um esquema de normalização (não usar, primeira normalização, segunda normalização, terceira normalização, quarta normalização); número de palavras usadas para formar uma assinatura digital; e granularidade. Para explorar as possíveis configurações, foram propostas 125 configurações.

Existe uma dificuldade em encontrar uma base de dados de códigos desenvolvidos por alunos com a classificação (plágio/não plágio; similar/não similar) (Đurić e Gašević [62]). Buscar uma boa configuração para o programa mostrou a necessidade desta base de códigos já previamente classificada. Isso impacta na validação.

Houve a necessidade de gerar manualmente uma base de dados de testes para o programa. Foram escolhidos doze programas de CS1 e para cada programa foram criadas

cinco duplas, formando sessenta duplas de códigos-fonte. Devido a ter 125 configurações e sessenta duplas, houve um total de 7.500 execuções do programa.

Os passos que seguem descrevem o procedimento manual usado para criar as cinco duplas para cada um dos 12 programas.

1. Escolher um código-fonte, etiquetá-lo como código-fonte padrão.
2. Escolher 2 implementações não similares entre si e também não similares com o código-fonte-padrão, etiquetá-las como implementacao1 e implementacao2.
3. Gerar 3 cópias do código-fonte-padrão e etiquetá-las como plagioA, plagioB, plagioC.
4. Alterar o plagioA, plagioB e plagioC visando simular ações feitas por alunos ao plagiar código-fonte.
   Ações plagioA: inclusão de comentários; mudança de nomes de variáveis; troca de posição de variáveis e funções.
   Ações plagioB: mudança de escopo de variável; alteração na indentação.
   Ações plagioC: inclusão de informação inútil (biblioteca, variável); rearranjo de expressão.
5. Gerar duplas de códigos-fonte para teste de análise de similaridade, sendo que cada dupla contém o código-fonte-padrão e um dos demais.
   Exemplo:
   Dupla1: código-fonte-padrão e plagioA;
   Dupla2: código-fonte-padrão e plagioB;
   Dupla3: código-fonte-padrão e plagioC;
   Dupla4: código-fonte-padrão e implementacao1;
   Dupla5: código-fonte-padrão e implementacao2;
6. Considerando cada dupla de código-fonte, etiquetá-la como plágio (1) e não plágio (0);
   Exemplo:
   Dupla1: código-fonte-padrão e plagioA; 1.
   Dupla2: código-fonte-padrão e plagioB; 1.
   Dupla3: código-fonte-padrão e plagioC; 1.
   Dupla4: código-fonte-padrão e implementacao1; 0.
   Dupla5: código-fonte-padrão e implementacao2; 0.
7. Executar a similaridade usando todo o conjunto de possíveis configurações para cada dupla de código-fonte.
8. Comparar o resultado da similaridade de cada dupla com os requisitos abaixo.
   Duplas etiquetadas como plágio (1): a similaridade precisa ser maior ou igual a 70%;
   Duplas etiquetadas como não plágio (0): a similaridade precisa ser menor que 70%.

9. Selecionar as configurações em que todas as ocorrências delas atendam aos requisitos do item 8.

A execução não trouxe nenhuma configuração que poderia ser usada nos 12 problemas. A configuração que poderia ser utilizada pelo maior número de problemas atende a sete problemas. Isso mostrou que em um ambiente real seria necessário encontrar qual configuração usar para cada problema, dificultando o processo. Para o usuário, esse cenário não é adequado.

A validação do *Sherlock NOverlap*, ao medir a qualidade das respostas, usou o método de conformidade confrontando resultados da ferramenta proposta com outras existentes. Ele argumenta essa estratégia por não se conhecer a priori a situação de pares de código como plágio ou não-plágio. Isso tem como objetivo aproximar a qualidade dos resultados do sistema proposto com os demais existentes, porém essa validação é questionável.

4.3 Estratégia Proposta

O desenvolvimento de um mecanismo para a detecção de plágio insere-se no contexto de um sistema de Juiz *Online* para facilitar seu uso em disciplinas de CS1. Por se tratar de um sistema voltado para a disciplina de CS1, os problemas definidos pelos professores são inicialmente simples, com soluções compostas por poucas linhas de código, como problemas com atribuições e expressões aritméticas, e finalizam em problemas um pouco mais difíceis, como envolvendo matrizes.

O banco de dados produzido por este projeto de pesquisa possui, atualmente, 100 problemas abordando os seguintes tópicos da disciplina: entrada de dados, saída de dados, estruturas de seleção, estruturas de repetição, cadeia de caracteres (*string*), estruturas de dados (vetores), uso de funções pré-definidas, definição de funções, passagem de parâmetros, domínio de lógica de programação, estrutura de dados (matriz), matemática. Esses problemas foram extensamente usados em sala de aula, gerando o banco de soluções usado para validar o sistema proposto. Atualmente o banco de dados possui 19.613 registros de soluções (programas submetidos).

Ao longo dos semestres de uso do *BOCA* como ferramenta de ensino, foi possível verificar que muitos alunos copiam as soluções que submetem. Ou da internet ou de outros alunos. Ao fazer o plágio, os alunos executam diversas ações, como: mudar a indentação do programa, mudar os nomes das variáveis, mudar o escopo das variáveis, adicionar variáveis, adicionar comentários, adicionar bibliotecas. A constatação da extensão em que o plágio é adotado pelos alunos motivou a inclusão de um mecanismo para detectar esta prática.

Houve a busca de um sistema já existente de detecção de plágio. Mas devido ao fato do único programa encontrado, *Sherlock N-Overlap*, segundo a literatura, com as características de (a) ter bons resultados, (b) ser *open source*, e (c) ter a possibilidade de ser executado localmente, ter sido testado para o contexto desta pesquisa sem apresentar bons resultados, tornou-se necessário desenvolver uma estratégia para resolver o problema.

Apesar da maioria dos sistemas identificados usarem o algoritmo *Greedy-String-Tiling*, que procura *substrings* que não se sobrepõem sem considerar a ordenação geral da *string* principal, nossa proposta, adaptada aos requisitos da disciplina de CS1, trabalha com o algoritmo Distância de Edição. O fato dessa disciplina ter programas menores influenciou na escolha do algoritmo Distância de Edição, pois ele não despreza a sequência do programa como um todo, como faz o *Greedy-String-Tiling*. Acredita-se que o algoritmo *Greedy-String-Tiling* seja mais útil para apoiar a identificação de plágio em programas maiores, porém é necessário realizar experimentos que comprovem esta suposição.

A estratégia proposta utiliza o algoritmo de Distância de Edição, *Levenshtein Distance* (Gilleland [30]), juntamente com a normalização feita no pré-processamento. O algoritmo Distância de Edição foi desenvolvido pelo cientista Russo Vladimir Levenshtein em 1965, e tem aplicação em verificação ortográfica, reconhecimento de fala, análise de DNA e detecção de plágio. Ele mede a distância entre duas *strings*, o que possibilita calcular a similaridade entre elas.

Para calcular a distância entre duas *strings*, o algoritmo faz uma contagem da quantidade de operações necessárias (inserção, deleção, modificação) para que uma *string* se iguale à outra [30]. A distância entre as *strings* "carro"e "cartas", é 3, porque para transformar "carro"em "cartas"é preciso fazer 2 modificações e 1 inserção. Já a distância entre as *strings* "computador"e "computador"é 0.

O algoritmo Distância de Edição é composto de sete passos, descritos no Algoritmo 4.1. O algoritmo recebe como argumento duas *strings* referenciadas como s e t.

Algoritmo 4.1: Passos do algoritmo Distância de Edição [30]

Entrada: s, t

Saída: d[n,m]

```
1  Passo 1: Defina n com o tamanho de s
2  Passo 1: Defina m com o tamanho de t
3  Passo 1: Se n=0 , retorne m e saia
4  Passo 1: Se m=0 , retorne n e saia
5  Passo 1: Construa uma matriz contendo m linhas e n colunas
6  Passo 2: Inicialize a primeira linha na posição 0 de n
7  Passo 2: Inicialize a primeira coluna na posição 0 de m
8  for Passo 3: Examine cada caractere de s (i de 1 a n) do
9      for Passo 4: Examine cada caractere de t (j de 1 a m) do
10         if Passo 5: s[i] == t[j] then
11             custo = 0
12         else
13             custo = 1
14         end
15         Passo 6: d[i,j] = menor(d[i-1,j] + 1, d[i,j-1] + 1, d[i-1,j-1] + custo)
16     end
17     Passo 7: A distância é o valor contido na célula d[n,m]
18 end
```

O passo 1 objetiva construir uma matriz cujo tamanho se refere ao tamanho das duas *strings* de entrada. O passo 2 posiciona um cursor na primeira posição da matriz para percorrer toda a matriz. Os passos 3 e 4 fazem com que o programa percorra todas as posições da matriz para preenchê-las conforme os passos que antecedem o passo 7. O passo 5, para cada iteração, faz comparações cruzadas entre o valor referente à posição da linha da matriz em uma das *strings* com o valor referente à posição da coluna da matriz na outra *string*. Após isso, define-se um valor denominado custo que se refere ao custo de alteração. O passo 6 preenche o valor da posição visitada com o menor valor entre as somas de cada um dos vizinhos (posição anterior, posição superior, posição na diagonal anterior) da posição visitada com o custo calculado no passo anterior, para a célula contida na diagonal, ou com um nos outros casos. Por fim o passo 7 retorna o valor da última posição da matriz como a distância entre as duas *strings*.

De maneira resumida, pode-se dizer que o Algoritmo 4.1 calcula o número de operações para transformar uma *string* na outra, para isso ele organiza os dados do processamento em uma matriz que acumula o resultado em uma perspectiva diagonal.

A figura 4.4 ilustra os passos percorridos pelo Algoritmo 4.1. O exemplo mostra a última célula com o valor referente ao número de operações necessárias para transformar a *string* "GUMBO"na *string* "GAMBOL", que implica na modificação do caractere "U"para "A"e da inserção do caractere "L"na primeira *string*.

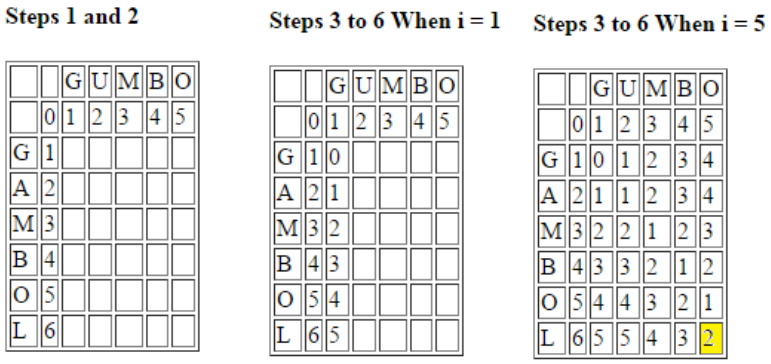

Figura 4.4: *Ilustração dos passos percorridos pelo algoritmo Distância de Edição [30]*

Uma implementação do algoritmo Distância de Edição com programação dinâmica [8] apresenta a complexidade de tempo de O(|s1| . |s2|), sendo *s1* e *s2* as duas *strings* de entrada. Ao considerar que as *strings* sejam de tamanhos iguais, a título de análise, a complexidade de tempo é O(n^2). As complexidades de espaço e tempo são iguais, porém é possível implementar o algoritmo reduzindo a complexidade de espaço para O(|s1|) ou O(n).

Visto que o Juiz *Online BOCA* foi desenvolvido em *PHP*, isso motivou o uso da implementação do algoritmo Distância de Edição disponível no endereço `http://php.net/manual/pt_BR/function.levenshtein.php`.

O algoritmo recebe como entrada duas *strings* normalizadas geradas a partir de dois arquivos de texto e retorna o número de operações necessárias para transformar uma *string* na outra, i.e a distância. Como a detecção de plágio tenta verificar a similaridade entre os códigos, considera-se que quanto menor for a distância entre o par de *strings* mais eles são similares. Neste contexto foi necessário criar uma equação para transformar a distância retornada pelo algoritmo para um percentual de similaridade.

$$maiorSubmissao = max(tamanhoSubmis1, tamanhoSubmis2) \tag{4-1}$$

$$similaridade = 100 - (\frac{distancia}{maiorSubmissao} * 100) \tag{4-2}$$

A variável *similaridade* recebe a transformação do valor retornado pela Distância de Edição entre as duas *strings* em um percentual de similaridade. Para isso, é utilizado o

quociente da Distância de Edição pelo número de caracteres da maior *string* normalizada, armazenada em *maiorSubmissao*.

Como mencionado anteriormente, o algoritmo recebe strings normalizadas. A normalização visa remover dos códigos-fonte informações irrelevantes quanto ao plágio e padronizá-los. Para isso foram removidos: declarações de bibliotecas, comentários, caracteres entre aspas e espaços em branco.

Inicialmente também tentou-se remover identificadores das variáveis como parte do pré-processamento. Essa ação foi implementada e os testes não trouxeram bons resultados, fazendo com que os programas ficassem naturalmente mais parecidos e acabou gerando muitos falsos-positivos.

A experiência corrobora a ideia da forte influência do pré-processamento na análise de plágio [37, 41]. Kleiman [37] verificou a vulnerabilidade, no pré-processamento, quanto a remoção de partes do programa, que se não for bem planejada pode remover partes importantes ou deixar de descartar partes insignificantes.

Essa influência do pré-processamento, associada ao fato de que a disciplina de CS1 trabalha muito com problemas simples e que os alunos que fazem plágio dificilmente teriam conhecimento suficiente para fazer modificações estruturais avançadas foram levadas em consideração na proposta apresentada, definindo as ações levadas a cabo no pré-processamento.

A Tabela 4.7 faz uma comparação da estratégia proposta com as demais, apresentadas na seção anterior. A coluna "Pré-processamento" é associada com os dados da Tabela 4.4. Percebe-se que a proposta deste trabalho não detecta mudanças estruturais e tem um pré-processamento simplificado. Essa simplificação foi adotada para tratar a vulnerabilidade apresentada por Kleiman [37].

4.3.1 Coleta de Dados

A estratégia de análise de similaridade projetada foi testada a fim de validar o trabalho feito. O algoritmo de similaridade foi executado para todos os pares de submissões corretas no banco de problemas do sistema *BOCA*.

Os problemas são organizados no banco de dados por assuntos e níveis de dificuldade. Foram criados grupos de assuntos para permitir identificar a qual grupo cada problema pertence. O grupo 1 possui os seguintes assuntos: entrada de dados, saída de dados, domínio de lógica de programação, e matemática. O grupo 2 possui estruturas de seleção. O grupo 3 possui estruturas de repetição. O grupo 4 possui: estruturas de dados (vetores), e cadeia de caracteres (*string*). O grupo 5 possui: uso de funções pré-definidas, definição de funções, passagem de parâmetros, e estrutura de dados (matriz). A dificuldade foi definida sendo de 1, fácil, a 3, difícil.

Tabela 4.7: *Comparação de estratégias usadas nos programas para análise de similaridade com a proposta.*

PROGRAMA	DETECTA MUDANÇAS LÉXICAIS	DETECTA MUDANÇAS ESTRUTURAIS	PRÉ-PROCESSAMENTO
CodeMatch	SIM	NÃO	1, 2
CPD	SIM	NÃO	3
Jplag	SIM	NÃO	1, 4, 5, 6
Plaggie	SIM	NÃO	1, 4, 5, 6
Marble	SIM	SIM	1, 4, 7, 8, 9, 10
MOSS	SIM	NÃO	* Informação Indisponivel
Sherlock	SIM	NÃO	* Informação Indisponivel
Sherlock N-Overlap	SIM	NÃO	1, 4, 5, 8, 11, 12, 13, 14.
SIM	SIM	NÃO	* Informação Indisponivel
YAP3	SIM	SIM	1, 16, 17, 18, 19, 15
SID	SIM	NÃO	3
ESTA PROPOSTA	SIM	NÃO	1, 4, 8, 7

Para comparar os resultados obtidos pelo algoritmo com uma análise humana a fim de verificar a eficácia da proposta, foram feitos 2 testes. O primeiro teste foi feito com 50 duplas de submissões, sendo 10 duplas escolhidas de maneira aleatória para cada um dos 5 grupos de assuntos. A classificação manual resultou em 24 duplas similares e 26 não-similares.

O segundo teste foi feito com 200 duplas divididas por problemas. Foram escolhidos 2 problemas de cada grupo, sendo um de menor nível de dificuldade e outro de maior nível. Para cada um dos 10 problemas foram selecionadas aleatoriamente 20 duplas, sendo 10 com similaridade menor ou igual à 85% e as outras 10 com similaridade maior que 85%, totalizando 100 duplas similares e 100 não similares. Para esse teste também foi feita uma classificação de plágio manualmente para cada registro selecionado e os resultados foram comparados.

4.3.2 Processamento

A análise de similaridade foi executada para pares de soluções corretas para cada problema. O processamento gerou 596.234 registros, sendo que cada um contém o par de arquivos analisados e seu respectivo grau de similaridade.

4.3.3 Resultados

Os resultados dos testes, na Tabela 4.8, são divididos em número de acertos (AC), número de falsos negativos (FN) e número de falsos positivos (FP). Os falsos negativos são os casos de plágio que o sistema classifica de maneira errada como não sendo plágio. Já os falsos positivos são os casos que não são plágio e o sistema classifica-os como plágio. Os falsos positivos são críticos, pois podem levar o professor a cometer injustiças com os alunos.

Tabela 4.8: *Testes da estratégia proposta com dados reais classificados manualmente*

	AC	AC %	FN	FN %	FP	FP %
Experimento com 50 duplas (> 75%)	46	92,00%	3	6,00%	1	2,00%
Experimento com 50 duplas (> 80%)	45	90,00%	4	8,00%	1	2,00%
Experimento com 50 duplas (> 85%)	44	88,00%	5	10,00%	1	2,00%
Experimento com 50 duplas (> 90%)	44	88,00%	6	12,00%	0	0,00%
Experimento com 50 duplas (> 95%)	44	88,00%	6	12,00%	0	0,00%
Experimento com 200 duplas (> 75%)	187	93,50%	6	3,00%	7	3,50%
Experimento com 200 duplas (> 80%)	188	94,00%	7	3,50%	5	2,50%
Experimento com 200 duplas (> 85%)	187	93,50%	8	4,00%	5	2,50%
Experimento com 200 duplas (> 90%)	171	85,50%	28	14,00%	1	0,50%
Experimento com 200 duplas (> 95%)	163	81,50%	37	18,50%	0	0,00%

A Tabela 4.8 mostra os resultados dos testes realizados considerando diferentes limiares para a classificação do plágio, variando de 75% a 95%. O limiar adotado, 85%, priorizou o balanceamento do número de acertos e do número de falsos-positivos. Observa-se que, conforme o limiar da similaridade diminui, o número de falsos-positivos aumenta. Conforme o limiar cresce, o número de falsos-positivos tende a reduzir, porém quando o limiar é maior, 80% no caso do experimento com 200 duplas, o número de acertos tende a diminuir. No sentido de obter um compromisso entre aumentar o número de acertos e reduzir os falsos-positivos, adotou-se o limiar de 85%, que não traz os melhores resultados, mas é um valor adequado por ser um pouco acima dos que trazem os melhores resultados, assim, reduzindo os falsos-positivos. Esse limiar pode ter que ser reavaliado quando o sistema for colocado em uso efetivo, sem esquecer que o sistema tem como objetivo servir como um apoio na detecção de plágio continuando a exigir uma análise por parte do professor, já que não consegue determinar com total certeza se ocorreu plágio ou não.

Assim, talvez o mais interessante não seja uma análise pontual de plágio, mas uma tendência de plágio pelo aluno ou pela turma. Aquele aluno ou turma que obtém resultados que indicam plágio na maioria dos exercícios de uma lista merece a atenção do professor para verificar a real situação.

4.4 Análise de Plágio no Contexto de uma Turma

Muitas pesquisas na área de informática e educação têm o interesse em propor ferramentas que possam apoiar o trabalho do professor visando melhoria no ensino. As ferramentas, além de serem capazes de automatizar diversos processos relacionados aos conteúdos e atividades, podem gerar dados para que o docente consiga obter conhecimento e tome as decisões mais acertadas visando qualidade. Essa linha de trabalho, aliada à questão do plágio no ensino de CS1, encoraja a levantar dados que relacionem os resultados obtidos pela ferramenta proposta e informações sobre turmas de alunos.

Para isso, foi desenvolvido um relatório que resume as informações sobre possibilidades de plágio para alunos de turmas especificas, i.e. dada a turma o relatório é gerado. Em cada linha são apresentados os dados dos alunos (nome, curso, ano, semestre, aprovação, nota final) e a relação de problemas resolvidos contendo: a identificação do problema, número de submissões, número de acertos, grau de similaridade. O "grau de similaridade" contém o maior índice de similaridade da solução correta submetida pelo aluno comparada às demais soluções oriundas da mesma turma.

No fim de cada linha é feito um resumo por aluno que apresenta: número de submissões, número de acertos, média de acertos por submissões, menor similaridade encontrada, similaridade média, maior similaridade encontrada, número de submissões com similaridade <= a 85%, e número de submissões com similaridade > que 85%. Na base do relatório, após apresentar todas as linhas para o conjunto de alunos, é apresentado o mesmo resumo por problema da lista. A intenção desse relatório é oferecer ao professor uma visão do comportamento dos alunos sobre plágio a partir de uma análise detalhada dos dados.

Além do benefício aos professores, essa abordagem pode ser interessante para pesquisadores que querem analisar o comportamento dos alunos quanto ao plágio. Duas análises desse tipo foram feitas com os dados disponíveis.

A Tabela 4.9 apresenta os dados de quatro turmas: número de alunos, representação das submissões (número de submissões, número de acertos, percentual de acertos por submissões), e representação da similaridade (número de submissões com <= 85% de similaridade, número de submissões com > 85% de similaridade, e percentual de submissões com > 85% de similaridade por acertos).

O campo "percentual de submissões com > 85% de similaridade por acertos" reflete o total de submissões corretas com um grau de similaridade que sugere plágio sobre o total de submissões corretas, visando representar o índice de plágio da turma. Ao ordenar os registros da tabela por esse campo obtém-se a sequência de turmas (A, C, B, D), sugerindo que houve mais casos de plágio na turma D, o que reflete a avaliação informal do professor.

Tabela 4.9: *Resumo de similaridade por turma considerando aspectos de submissões.*

Turma	Número de alunos	Número de Submissões	Número de Acertos	de acertos por submissões	Submissões com <= 85% de similaridade	Submissões com > 85% de similaridade	Submissões com >85% de similaridade por acertos
A	44	3276	1619	49,42%	924	731	45,15%
B	37	5316	3605	67,81%	1459	2175	60,33%
C	42	3210	1623	50,56%	680	964	59,40%
D	22	1955	1346	68,85%	339	1021	75,85%

Já o campo "percentual de acertos por submissões" representa para o conjunto de todas as submissões o tamanho do subconjunto de submissões corretas representado por um percentual. Esse campo foi criado a partir da constatação de que, geralmente, os alunos que respondem os problemas sem fazer plágio costumam errar nas primeiras tentativas e após isso enviam a resposta correta. Nessa analogia, alto percentual indica maior possibilidade de plágio. Ao ordenar as turmas por esse campo, obtém-se (A, C, B, D).

Observa-se que o mesmo conjunto ordenado de turmas, (A, C, B, D), foi encontrado pelos dois campos analisados. Isso mostrou, para este conjunto de dados, que quando os alunos enviaram poucas submissões para cada problema e mesmo assim conseguiram acertá-lo, a possibilidade de estarem plagiando cresceu.

Também foi feita uma análise da correlação entre a nota do aluno e tendência de plágio, para entender se existe alguma relação entre plágio e notas ruins na disciplina. Para isso, calculou-se o número de submissões com similaridade > 85% proporcional ao número de acertos do aluno. Mais uma vez essa divisão pelo número de acertos foi para nivelar a quantidade de submissões tendenciosas a plágio pela participação do aluno considerando seus acertos.

Resultados relativos ao Coeficiente de Correlação de *Pearson* (Tabela 4.10), que foram usados para verificar a direção da correlação entre as variáveis, mostram que há uma correlação negativa significativa entre o "número de submissões com similaridade > 85% proporcional ao número de acertos do aluno" e as notas dos alunos nas turmas A, B e C. Assim, maior "número de submissões com similaridade > 85% proporcional ao número de acertos do aluno" está associada a menor nota.

Tabela 4.10: *Correlações de notas com submissões similaridade para turmas.*

TURMA	*r*	*p*
A	-.44	.003
B	-.35	.035
C	-.27	.082
D	.17	.462

A turma D apresentou uma correlação positiva não significativa, o que permite concluir que não há associação entre as variáveis nesse caso. Isso quer dizer que não se verificou correlação entre plágio e nota. Uma análise mais detalhada do contexto teria que ser feita para entender esta diferença de resultados. É importante lembrar que existem vários fatores que podem influenciar esse resultado, e.g. os alunos podem "colar" durante as provas e o professor não consegue identificar, os exercícios podem não estar relacionados à prova, a nota final pode ter sido obtida por um cálculo que não leva em consideração os exercícios, entre outros.

Envolvendo plágio e o comportamento dos alunos, as informações obtidas são importantes e devem ser consideradas nos requisitos de sistemas de apoio ao ensino de CS1. Elas contribuem como apoio ao corpo docente por meio de alertas sobre o comportamento dos alunos. Além disso, manter um registro dessas informações permite criar um banco de conhecimento para a continuação de estudos científicos nesta área.

CAPÍTULO 5

Conclusão

A complexidade da aprendizagem de programação, associada à falta do conhecimento prévio, como a baixa habilidade raciocínio matemático e a dificuldade de interpretação de problemas, tornam desafiadora a tarefa das Universidades que se propõem a ministrar cursos na área de computação.

A literatura mostra que esse é um problema recorrente em todo o mundo. Diversas estratégias pedagógicas, incluindo ferramentas e metodologias, têm sido propostas mas ainda não se tem um consenso sobre qual a melhor solução. No entanto, alguns problemas são recorrentes e devem ser tratados. A necessidade do aluno resolver um grande número de problemas exige um grande esforço do professor na elaboração e correção das listas de exercícios. As ferramentas de Juiz *Online* contribuem fortemente para o ensino de diversas disciplinas que envolvem programação de computadores, pois reduzem o peso do trabalho do professor e permitem que ele se preocupe com outras questões necessárias para um ensino adequado a cada realidade. Ainda mais, há um alto índice de plágio na disciplina de CS1 comprovado em diferentes referências. As pesquisas apontam diversas questões sobre o plágio, como o fato do aluno necessitar da obtenção da nota e a formalidade das linguagens de programação usadas para resolver problemas bem delimitados que não oferecem grande margem de se fazer soluções diferentes. O uso de ferramentas de Juiz *Online* pode contribuir no plágio, ao facilitar a cópia e envio de soluções, mas por outro lado pode ajudar na identificação do plágio ao permitir comparação automática de soluções.

Esta pesquisa iniciou com o objetivo de explorar o Juiz *Online BOCA* no ensino de CS1 para facilitar a geração de listas de problemas e identificar os problemas relacionados. Para isso usou-se o sistema em disciplinas de introdução à programação na UFG e no IFGoiano, o que resultou em um BD de interações de alunos com o sistema e experiências para a condução da pesquisa. O uso na prática permitiu o levantamento dos requisitos importantes para um sistema de Juiz *Online* para o ensino de CS1. Entre esses requisitos, a definição de uma estratégia para medir a dificuldade de problemas, que contribui para a geração de listas de problemas, e a identificação de plágio se destacam pelo desafio que oferecem. Assim, neste trabalho, esses aspectos foram focados.

A realização da RSL sobre Juiz *Online* no ensino de CS1 e as experiências realizadas na UFG e no IFGoiano foram fundamentais para a descrição dos requisitos do sistema proposto, sendo classificados em quatro conjuntos de requisitos funcionais (*feedback*, integração do sistema com os cursos, análise do desempenho geral dos alunos, e oferecer diferentes tipos de atividades) e cinco requisitos não-funcionais (integração, usabilidade, segurança, escalabilidade, disponibilidade). A experiência mostrou benefícios quanto ao aumento da produtividade no trabalho do professor e problemas relacionados à ocorrência de plágio e à necessidade de modificações em seu código-fonte para adaptações. Como resultado, propõe-se um sistema flexível no sentido de possibilitar aos docentes configurarem o que desejam usar, permitindo um vínculo com sua realidade de ensino. Quanto ao sistema mais adequado à disciplina de CS1, ainda não existe uma validação consistente sobre a contribuição do uso de sistemas de Juiz *Online* na aprendizagem do aluno, apesar do benefício considerável na eficiência do trabalho do professor. A definição de uma estratégia de validação é apresentada pela literatura de educação em computação como fator crítico e que precisa de ser enfatizado nas pesquisas para a evolução da área.

Para medir a dificuldade de problemas de CS1, e contribuir para a geração de listas de problemas no contexto de Juízes *Online*, foram propostas diversas equações que foram testadas usando o banco de soluções obtido com o uso do *BOCA*. Coeficientes de Correlação de *Pearson* foram calculados e mostraram resultados significantes na equação que usa o número de assuntos abordados no problema e a altura de uma árvore em que cada subconjunto de códigos aninhados internos representa um nó. A altura dessa árvore foi usada para explorar a estrutura dos algoritmos, que puderam ser representados por árvore e grafo, visto que o algoritmo representa semanticamente o problema e existe uma inclinação na literatura em tratar a dificuldade de compreensão textual na resolução de problemas de CS1. Esta abordagem não considerou a capacidade cognitiva do aluno no momento em que ele classifica a dificuldade do problema, o que é importante para investigar mais afundo, devido a não se dispor de tais dados, porém foram obtidos resultados satisfatórios que contribuem para a geração de listas de problemas no contexto trabalhado.

Para apoiar o docente na identificação de plágio em código-fonte de atividades de CS1 submetidas ao sistema *BOCA*, uma estratégia foi projetada com o algoritmo Distância de Edição, que mede a distância entre duas *strings*, e técnicas de pré-processamento. O uso do algoritmo Distância de Edição e não o algoritmo *Greedy-String-Tiling*, frequentemente mencionado na literatura, deve-se ao fato que o algoritmo Distância de Edição preserva a ordenação dos caracteres das *strings* comparadas como um todo, o que é importante já que o conteúdo de CS1 possui um conjunto de comandos muito limitado e os programas tendem a ser pequenos. O algoritmo *Greedy-String-Tiling* procura *substrings* que não se sobrepõem, criando sub-conjuntos de *strings*, sendo que a preservação da or-

denação é limitada a cada sub-conjunto. A estratégia de pré-processamento proposta visa criar um algoritmo que balanceasse a quantidade de informação removida dos códigos fonte para manter a identidade dos códigos e, ao mesmo tempo, permitir a comparação. A normalização proposta remove declarações de bibliotecas, comentários, caracteres entre aspas e espaços em branco.

É uma solução de fácil implementação, o que permite seu uso em sistemas com os mesmos requisitos sem dificuldade. Não é necessário fazer configurações específicas para cada problema, como é feito no *Sherlock N-Overlap*, pois trata-se de um mecanismo único para usar em diferentes problemas da disciplina em questão. Para validar a proposta foram realizados testes usando submissões reais, oriundas de interações de alunos com o Juiz *Online BOCA*. Conjuntos de pares dessas submissões foram analisados pelos pesquisadores e classificadas de acordo com sua percepção da existência de plágio ou não.

Os resultados obtidos ao aplicar a estratégia de identificação de plágio no banco de dados possibilitaram fazer análises em um contexto mais amplo a partir da delimitação de turmas. Foi possível mostrar o índice geral de plágio de um aluno comparando às submissões enviadas pelos demais alunos de sua turma, que foi obtido a partir do índice de similaridade individual de cada problema resolvido, e um percentual que mostra a inclinação da turma para realizar o plágio. Analisar esses dados incluindo notas de alunos mostrou casos em que há uma tendência para alunos plagiadores em serem reprovados na disciplina. Realizar essas análises no decorrer do tempo contribui para que a equipe docente possa tomar as decisões mais adequadas a tempo hábil, visando corrigir o problema já no início do curso.

As principais dificuldades enfrentadas por esta pesquisa foram (a) necessitar de dados subjetivos para as validações, pois propor uma estratégia para calcular a dificuldade de problemas simulando a visão dos alunos depende de dados subjetivos dos alunos e propor uma estratégia para classificar plágio depende de dados subjetivos dos professores, e (b) entender o código-fonte do sistema *BOCA* para realizar manutenções de *software*. O sistema *BOCA* apresentou dificuldades na realização de manutenções.

Ao trabalhar com a computação aplicada, é necessário realizar a validação conforme a área fim (neste caso a educação). Considerando a informática na educação, em muitos casos, é necessário verificar o impacto do uso de um sistema na aprendizagem do aluno. A forma mais adequada de se fazer isso é melhor definida por um profissional da área fim. Além disso, várias outras questões devem ser tratadas, como o plágio interferindo nos dados, a subjetividade dos alunos, as questões éticas para lidar com dados de pessoas, etc. Assim, é importante trabalhar com equipes multidisciplinares, com indivíduos com papeis bem definidos, como estatísticos, psicólogos, pedagogos, e profissionais da área de computação, permitindo uma correta conexão entre as áreas.

O trabalho gerou como resultado três artigos. Foi publicado no *International Conference on Educational Data Mining - EDM 2015* um artigo abordando a dificuldade dos problemas de CS1 no contexto do Juiz *Online* BOCA [26]. O segundo artigo, sobre a estratégia proposta para apoiar na identificação de plágio em CS1 [27], foi publicado na revista *iSys - Revista Brasileira de Sistemas de Informação*. O terceiro artigo, que aborda a RSL sobre Juiz *Online* e ensino de CS1, foi aceito para publicação no *XXVII Simpósio Brasileiro de Informática na Educação*, evento parte do *V Congresso Brasileiro de Informática na Educação*.

5.1 Trabalhos Futuros

A realização desta pesquisa focou dois aspectos importantes de um sistema de Juiz Online mas apontou vários outros que ainda devem ser tratados. Uma proposta geral do sistema foi apresentada. Sugere-se um projeto de sistema de Juiz *Online* que seja (1) flexível no sentido de permitir ao professor configurar como quer que o sistema funcione, oferecendo entre as funcionalidades as propostas por este trabalho, (2) considere padrões de engenharia de *software* e tenha o código-fonte aberto para permitir sua evolução e comunicação entre pesquisadores de diferentes Universidades, e (3) possa ser usado livremente por estudantes e professores, para isso precisará de considerar requisitos de escalabilidade e segurança. Propõem-se implementar este sistema integrando os módulos trabalhados nesta dissertação.

É interessante trabalhar com Juízes *Online* visando a coleta, o armazenamento, e a análise de dados das interações entre alunos e problemas em diferentes contextos de ensino. Trabalhar com questões relacionadas à anonimizações e políticas de segurança dos dados é importante para permitir disponibilizar esse BD para diferentes pesquisadores. A ciência dos dados é promissora, permitindo obter novos conhecimento sobre a aprendizagem de programação de computadores.

Avaliar a qualidade de sistemas educacionais é um desafio. É interessante que sejam feitas pesquisas que foquem exclusivamente na validação de sistemas para o ensino considerando os vários contextos educacionais. Trabalhar com notas dos alunos e verificar se os alunos concordam que o sistema contribuiu para sua aprendizagem são exemplos de informações que podem ser levantadas. No entanto é importante que se use um método estatístico que permita a comparação de diversos contextos e, inclusive, sirva como referência para pesquisas de qualidade.

Todos os trabalhos retornados pela RSL sobre dificuldade de problemas de CS1 [15, 19, 25, 39, 40, 59] tiveram problemas ao validar suas propostas. Denny et. al. [25] consideraram que estimar a dificuldade de um problema necessita da consideração de quais conceitos foram apresentados para os alunos até o momento. Isso impacta

diretamente na validação das pesquisas, trazendo a possibilidade da criação de conjuntos de alunos com capacidades semelhantes quanto ao desenvolvimento na disciplina e sua sua relação com a dificuldade dos problemas.

A dificuldade relacionada à resolução do problema, que envolve sua compreensão e capacidade de raciocínio lógico, foi discutida nos trabalhos [15, 19, 25, 39, 44, 59]. Sugere-se como trabalhos futuros abordar isso com outras estratégias, como considerando a descrição textual e o contexto do problema com o uso de tecnologias semânticas.

Por fim, propõe-se abordar às ameaças [7, 28] às validações das estratégias para apoiar à identificação de plágio. Gaudencio et. al. [28] mostraram a discordância entre professores ao classificarem a similaridade entre programas de CS1 e Al-Khanjari et. al. [7] mostraram a necessidade de estudar a linha tênue entre os níveis aceitável e inaceitável de semelhanças entre os programas, o qual eles acreditam depender do ambiente de programação. É necessário chegar a um consenso sobre essa discordância na classificação do plágio para tornar os dados mais objetivos e confiáveis e estudar estatisticamente como essa linha tênue se dá ao comparar vários experimentos para um mesmo contexto.

Referências Bibliográficas

[1] **ACM. ACM-ICP Live Archive. Disponível em https://icpcarchive.ecs.baylor.edu. Acessado em 20 de maio de 2014.**

[2] **Sherlock. The Sherlock Plagiarism Detector. Disponível em http://sydney.edu.au/engineering/it/ scilect/sherlock/. Acessado em 28 de junho de 2015.**

[3] **SPOJ. SPOJ Brasil. Disponível em http://br.spoj.com/embed/info/. Acessado em 20 de maio de 2014.**

[4] **URI. URI Online Judge. Disponível em http://www.urionlinejudge.com.br/. Acessado em 20 de maio de 2014.**

[5] **UVA. UVa Online Judge. Disponível em http://uva.onlinejudge.org/. Acessado em 20 de maio de 2014.**

[6] AHTIAINEN, A.; SURAKKA, S.; RAHIKAINEN, M. **Plaggie: Gnu-licensed source code plagiarism detection engine for java exercises**. In: *Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006*, p. 141–142. ACM, 2006.

[7] AL-KHANJARI, Z. A.; FIAIDHI, J.; AL-HINAI, R.; KUTTI, N. S. **Plagdetect: a java programming plagiarism detection tool**. *ACM Inroads*, 1(4):66–71, 2010.

[8] ALISSON, L. **Dynamic programming algorithm (dpa) for edit-distance**. http://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Edit [Acessado em 28 de dezembro de 2015].

[9] ALSHAMSI, F.; ELNAGAR, A. **An automated assessment and reporting tool for introductory java programs**. In: *Innovations in Information Technology (IIT), 2011 International Conference on*, p. 324–329. IEEE, 2011.

[10] ALVAREZ, A.; SCOTT, T. A. **Using student surveys in determining the difficulty of programming assignments**. *Journal of Computing Sciences in Colleges*, 26(2):157–163, 2010.

[11] AMBRÓSIO, A. P. L.; ALMEIDA, L. S.; MACEDO, J.; SANTOS, A.; FRANCO, A. H. R. **Programação de computadores: compreender as dificuldades de aprendizagem dos alunos**. 2011.

[12] BAKSHI, T. **Computer science tops in academic violations last year. the brown daily herald. 1 november 2010.**, 2010. http://www.browndailyherald.com/mobile/computer-science-tops-in-academic-violations-last-year-1.2388353 [Acessado em 20 de janeiro de 2016].

[13] BAUGH, J.; KOVACS, P.; DAVIS, G. **Does the computer programming student understand what constitutes plagiarism**. *Issues in Information Systems*, 13(2):138–145, 2012.

[14] BEZERRA, E. **Princípios De Análise E Projeto De Sistemas Com Uml-3ª Edição**, volume 3. Elsevier Brasil, 2015.

[15] BRYCE, R. C.; COOLEY, A.; HANSEN, A.; HAYRAPETYAN, N. **A one year empirical study of student programming bugs**. In: *Frontiers in Education Conference (FIE), 2010 IEEE*, p. F1G–1. IEEE, 2010.

[16] BYRNE, P.; LYONS, G. **The effect of student attributes on success in programming**. In: *ACM SIGCSE Bulletin*, volume 33, p. 49–52. ACM, 2001.

[17] CHAVES, J. O.; CASTRO, A.; LIMA, R.; LIMA, M. V.; FERREIRA, K.; FAÇANHA., A. **Uma ferramenta de auxílio ao processo de ensino aprendizagem para disciplinas de programação de computadores**. *TISE 2013*, 2013.

[18] CHEN, X.; FRANCIA, B.; LI, M.; MCKINNON, B.; SEKER, A. **Shared information and program plagiarism detection**. *Information Theory, IEEE Transactions on*, 50(7):1545–1551, 2004.

[19] CHERENKOVA, Y.; ZINGARO, D.; PETERSEN, A. **Identifying challenging cs1 concepts in a large problem dataset**. In: *Proceedings of the 45th ACM technical symposium on Computer science education*, p. 695–700. ACM, 2014.

[20] CHMURA, G. A. **What abilities are necessary for success in computer science**. *ACM SIGCSE Bulletin*, 30(4):55–58, 1998.

[21] COPELAND, T. **Detecting duplicate code with pmds cpd.** 2003.

[22] COSTA[1], T. H.; BUBLITZ[1], F. M. **Análise dos principais problemas que afetam alunos de programação: uma investigação empírica no estado da paraíba.** 2013.

[23] DE CAMPOS, C. P.; FERREIRA, C. E. **Boca: um sistema de apoio a competições de programação**. In: *Workshop de Educação em Computação*, p. 885–895, 2004.

[24] DEHNADI, S. **A cognitive study of learning to program in introductory programming courses**. PhD thesis, Middlesex University, 2009.

[25] DENNY, P.; CUKIERMAN, D.; BHASKAR, J. **Measuring the effect of inventing practice exercises on learning in an introductory programming course**. In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, p. 13–22. ACM, 2015.

[26] FRANCISCO, R. E.; AMBROSIO, A. P. **Mining an online judge system to support introductory computer programming teaching**. *SMLIR: Workshop on Tools and Technologies in Statistics, Machine Learning and Information Retrieval for Educational Data Mining*, 2015.

[27] FRANCISCO, R. E.; AMBRÓSIO, A. P. L. **Uso do algoritmo distância de edição com técnicas de pré-processamento para apoiar a identificação de plágio em códigos-fonte de problemas de programação introdutória**. *iSys-Revista Brasileira de Sistemas de Informação*, 9(2), 2016.

[28] GAUDENCIO, M.; DANTAS, A.; GUERRERO, D. D. **Can computers compare student code solutions as well as teachers?** In: *Proceedings of the 45th ACM technical symposium on Computer science education*, p. 21–26. ACM, 2014.

[29] GEORGOULI, K.; GUERREIRO, P. **Incorporating an automatic judge into blended learning programming activities**. In: *Advances in Web-Based Learning–ICWL 2010*, p. 81–90. Springer, 2010.

[30] GILLELAND, M. **Levenshtein distance, in three flavors. merriam park software**, 2006. http://people.cs.pitt.edu/~kirk/cs1501/Pruhs/Fall2006/Assignments/editdistance/Levenshtein%20Distance.htm [Acessado em 28 de dezembro de 2015].

[31] GOMES, A.; AREIAS, C.; HENRIQUES, J.; MENDES, A. J. **Aprendizagem de programação de computadores: dificuldades e ferramentas de suporte**. *Revista Portuguesa de Pedagogia*, 42(2), 2008.

[32] GRUNE, D. **Sim(1)**, 2012. http://dickgrune.com/Programs/similarity_tester/sim.pdf [Acessado em 24 de janeiro de 2016].

[33] HAGE, J.; RADEMAKER, P.; VAN VUGT, N. **A comparison of plagiarism detection tools**. *Utrecht University. Utrecht, The Netherlands*, p. 28, 2010.

[34] HUITT, W.; HUMMEL, J. **Piaget's theory of cognitive development**. *Educational psychology interactive*, 3(2), 2003.

[35] JOY, M.; LUCK, M. **Plagiarism in programming assignments**. *Education, IEEE Transactions on*, 42(2):129–133, 1999.

[36] KITCHENHAM, B. **Procedures for performing systematic reviews**. *Keele, UK, Keele University*, 33(2004):1–26, 2004.

[37] KLEIMAN, A. B. **Análise e comparaçao qualitativa de sistemas de detecçao de plágio em tarefas de programaçao**. PhD thesis, Instituto de Computaçao, 2007.

[38] KRAMER, J. **Is abstraction the key to computing?** *Communications of the ACM*, 50(4):36–42, 2007.

[39] LAKANEN, A.-J.; LAPPALAINEN, V.; ISOMÖTTÖNEN, V. **Revisiting rainfall to explore exam questions and performance on cs1**. In: *Proceedings of the 15th Koli Calling Conference on Computing Education Research*, p. 40–49. ACM, 2015.

[40] LLANA, L.; MARTIN-MARTIN, E.; PAREJA-FLORES, C. **Flop, a free laboratory of programming**. In: *Proceedings of the 12th Koli Calling International Conference on Computing Education Research*, p. 93–99. ACM, 2012.

[41] MACIEL, D. L. **Sherlock n-overlap: Normalização invasiva e coeficiente de sobreposição para análise de similaridade entre códigos-fonte em disciplinas de programação**. 2014.

[42] MARTINS, S. W.; MENDES, A. J. **Comunidades de investigação em programação: Uma estratégia de apoio ao aprendizado inicial de programação**. *IEEE-RITA Vol. 5*, 2010.

[43] MARTINS, V. T.; FONTE, D.; HENRIQUES, P. R.; DA CRUZ, D. **Plagiarism detection: A tool survey and comparison**. *3rd Symposium on Languages, Applications and Technologies (SLATE 14)*., p. 4566, 2014.

[44] MENDONÇA, A.; DE OLIVEIRA, C.; GUERRERO, D.; COSTA, E. **Difficulties in solving ill-defined problems: A case study with introductory computer programming students**. In: *Frontiers in Education Conference, 2009. FIE'09. 39th IEEE*, p. 1–6. IEEE, 2009.

[45] MOTA, A. M.; GOYA, D. H. **Técnicas para análise de similaridade de código de software em litígios de propriedade intelecttual**.

[46] OHMANN, T.; RAHAL, I. **Efficient clustering-based source code plagiarism detection using piy**. *Knowledge and Information Systems*, 43(2):445–472, 2015.

[47] PADUELLI, M. M.; SANCHES, R. **Problemas em manutenção de software: caracterização e evolução**. *III Workshop de Manutenção de Software Moderna - WMSWM 2006*, 2006.

[48] PETIT, J.; GIMÉNEZ, O.; ROURA, S. **Jutge. org: an educational programming judge**. In: *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, p. 445–450. ACM, 2012.

[49] PETTIT, R.; HOMER, J.; HOLCOMB, K.; SIMONE, N.; MENGEL, S. **Are automated assessment tools helpful in programming courses?** *122nd ASEE Annual Conference and Exposition. American Society for Engineering Education, 2015.*, 2015.

[50] PETTIT, R.; HOMER, J.; GEE, R.; MENGEL, S.; STARBUCK, A. **An empirical study of iterative improvement in programming assignments**. In: *Proceedings of the 46th ACM Technical Symposium on Computer Science Education*, p. 410–415. ACM, 2015.

[51] POZENEL, M.; FURST, L.; MAHNICC, V. **Introduction of the automated assessment of homework assignments in a university-level programming course**. In: *Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015 38th International Convention on*, p. 761–766. IEEE, 2015.

[52] PRECHELT, L.; MALPOHL, G.; PHILIPPSEN, M. *J. UCS*, 8(11):1016, 2002.

[53] RAJALA, T.; KAILA, E.; LINDÉN, R.; KURVINEN, E.; LOKKILA, E.; LAAKSO, M.-J.; SALAKOSKI, T. **Automatically assessed electronic exams in programming courses**. In: *Proceedings of the Australasian Computer Science Week Multiconference*, p. 11. ACM, 2016.

[54] RODRIGUES, F. S. **Estudo sobre a evasão no curso de ciência da computação da ufrgs**. 2013.

[55] ROUNTREE, N.; ROUNTREE, J.; ROBINS, A.; HANNAH, R. **Interacting factors that predict success and failure in a cs1 course**. *ACM SIGCSE Bulletin*, 36(4):101–104, 2004.

[56] SAFE. **Codematch algorithms**. http://www.safe-corp.biz/CodeMatch_algorithms.htm [Acessado em 21 de janeiro de 2016].

[57] SCHLEIMER, S.; WILKERSON, D. S.; AIKEN, A. **Winnowing: local algorithms for document fingerprinting**. In: *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, p. 76–85. ACM, 2003.

[58] SHEARD, J.; DICK, M.; MARKHAM, S.; MACDONALD, I.; WALSH, M. **Cheating and plagiarism: perceptions and practices of first year it students**. In: *ACM SIGCSE Bulletin*, volume 34, p. 183–187. ACM, 2002.

[59] SIMON. **Assignment and sequence: Why some students can't recognise a simple swap**. *Koli Calling '11*, 2011.

[60] SOUZA, D. M. D.; ISOTANI, S.; BARBOSA, E. F. **Teaching novice programmers using progtest**. *International Journal of Knowledge and Learning*, 10(1):60–77, 2015.

[61] SUN, H.; LI, B.; JIAO, M. **Yoj: An online judge system designed for programming courses**. In: *Computer Science & Education (ICCSE), 2014 9th International Conference on*, p. 812–816. IEEE, 2014.

[62] ĐURIĆ, Z.; GAŠEVIĆ, D. **A source code similarity system for plagiarism detection**. *The Computer Journal*, p. bxs018, 2012.

[63] VIEIRA, M. A. **Modelagem de Espaços Inteligentes Pessoais e Espaços Inteligentes Fixos no contexto de Cenários de Computação Ubíqua - Dissertação de Mestrado em Ciência da Computação.** PhD thesis, Universidade Federal de Goiás, 2016.

[64] VIHAVAINEN, A.; LUUKKAINEN, M.; PÄRTEL, M. **Test my code: An automatic assessment service for the extreme apprenticeship method**. In: *2nd International Workshop on Evidence-based Technology Enhanced Learning*, p. 109–116. Springer, 2013.

[65] WAGNER, N. R. **Plagiarism by student programmers**, 2000. http://www.cs.utsa.edu/~wagner/pubs/plagiarism0.html [Acessado em 20 de janeiro de 2016].

[66] WANG, T.; SU, X.; MA, P.; WANG, Y.; WANG, K. **Ability-training-oriented automated assessment in introductory programming course**. *Computers & Education*, 56(1):220–226, 2011.

[67] WAZLAWICK, R. S. **Uma reflexão sobre a pesquisa em ciência da computação à luz da classificação das ciências e do método científico**. *Revista de Sistemas de Informação da FSMA*, 6:3–10, 2010.

[68] WILSON, B. C.; SHROCK, S. **Contributing to success in an introductory computer science course: a study of twelve factors**. In: *ACM SIGCSE Bulletin*, volume 33, p. 184–188. ACM, 2001.

[69] WISE, M. J. **Yap3: improved detection of similarities in computer program and other texts**. In: *ACM SIGCSE Bulletin*, volume 28, p. 130–134. ACM, 1996.

[70] ZEIDMAN, R. **Software source code correlation**. In: *Computer and Information Science, 2006 and 2006 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse. ICIS-COMSAR 2006. 5th IEEE/ACIS International Conference on*, p. 383–392. IEEE, 2006.

APÊNDICE A

Protocolo da Revisão Sistemática da Literatura sobre Juiz Online no ensino de CS1

O modelo de protocolo para Revisão Sistemática da Literatura usado para a condução desta pesquisa foi adaptado de Kitchenham [36]. Neste protocolo, são apresentadas as decisões técnicas sobre a condução da pesquisa, que inclui desde a criação da *string* de busca até a definição de critérios que decidem o uso de cada trabalho encontrado.

- **Questão principal**
  Qual é a especificação mais adequada para que um juiz online atenda à disciplina de CS1?

- **Questões de pesquisa**
  -QP1: Quais os benefícios ao usar juiz online no ensino de CS1?
  -QP2: Quais os problemas ao usar juiz online no ensino de CS1?
  -QP3: Quais requisitos funcionais um juiz online precisa atender para ser usado no ensino de CS1?
  -QP4: Quais requisitos não-funcionais um juiz online precisa atender para ser usado no ensino de CS1?

- **População**
  Artigos científicos que envolvem juiz online e ensino de CS1.

- **Critério de seleção das bases de pesquisa**
  -Disponibilizar mecanismo de consulta via web.
  -Serem relacionadas a temas de Computação e Informática.
  -Permitir filtro por ano de publicação.

- **Bases de pesquisa selecionadas**
  - IEEEXplorer Digital Library
  - ACM Digital Library

- Springer
- Science Direct
- Scopus

- **Idiomas**
  Português e Inglês

- **Strings de busca**
  ("online judge"OR "automated assessment"OR "automatic assessment") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")

  ("juiz online"OR "avaliação automática"OR "correção automática") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

- **Filtro da busca**
  Publicações entre 2010 a 2016. Pesquisar a *string* nos campos Título e Resumo.

- **Critérios para exclusão**
  - E1: Texto completo não disponível para acesso gratuitamente na web ou no portal de periódicos da Capes.
  - E2: Escrito outro idioma.
  - E3: Trabalho não aborda juiz online e ensino de CS1.
  - E4: Não é um capítulo de livro com resumo ou artigo de periódico ou de conferência
  - E5: Publicação duplicada.

- **Critérios para inclusão**
  - I1: Aborda os benefícios ao usar juiz online no ensino de CS1.
  - I2: Aborda os problemas ao usar juiz online no ensino de CS1.
  - I3: Aborda os requisitos funcionais que um juiz online precisa atender para ser usado no ensino de CS1.
  - I4: Aborda os requisitos não-funcionais que um juiz online precisa atender para ser usado no ensino de CS1.

- **Fase de seleção**
  Os critérios de inclusão e exclusão serão aplicados considerando o título, palavras-

chave e resumo.

- **Fase de extração**
  Os critérios de inclusão e exclusão serão aplicados considerando o texto completo.

- **Strings adaptadas à cada base**
  IEEEXplorer Digital Library
  ("online judge"OR "automated assessment"OR "automatic assessment") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
  ("juiz online"OR "avaliação automática"OR "correção automática") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

  Springer
  ("online judge"OR "automated assessment"OR "automatic assessment") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
  ("juiz online"OR "avaliação automática"OR "avaliação automática") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

  Science Direct
  pub-date > 2009 and pub-date < 2017 and TITLE-ABSTR-KEY("online judge"OR "automated assessment"OR "automatic assessment") AND TITLE-ABSTR-KEY("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
  pub-date > 2009 and pub-date < 2017 and TITLE-ABSTR-KEY("juiz online"OR "avaliação automática"OR "avaliação automática") AND TITLE-ABSTR-KEY("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

  Scopus
  TITLE-ABS-KEY (("online judge"OR "automated assessment"OR "automatic assessment") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")) AND PUBYEAR > 2009 AND PUBYEAR < 2017
  TITLE-ABS-KEY (("juiz online"OR "avaliação automática"OR "avaliação au-

tomática") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")) AND PUBYEAR > 2009 AND PUBYEAR < 2017

ACM Digital Library

acmdlTitle:(+ "online judge"+ "introductory programming") OR recordAbstract:(+"online judge"+"introductory programming") OR keywords.author.keyword:(+"online judge"+ "introductory programming")

acmdlTitle:(+"online judge"+"introduction to programming") OR recordAbstract:(+"online judge"+"introduction to programming") OR keywords.author.keyword:(+"online judge"+"introduction to programming")

acmdlTitle:(+"online judge"+"computer science 1") OR recordAbstract:(+"online judge"+"computer science 1") OR keywords.author.keyword:(+"online judge"+"computer science 1")

acmdlTitle:(+"online judge"+"CS1") OR recordAbstract:(+"online judge"+"CS1") OR keywords.author.keyword:(+"online judge"+"CS1")

acmdlTitle:(+"automated assessment"+"introductory programming") OR recordAbstract:(+"automated assessment"+"introductory programming") OR keywords.author.keyword:(+"automated assessment"+"introductory programming")

acmdlTitle:(+"automated assessment"+"introduction to programming") OR recordAbstract:(+"automated assessment"+"introduction to programming") OR keywords.author.keyword:(+"automated assessment"+"introduction to programming")

acmdlTitle:(+"automated assessment"+"computer science 1") OR recordAbstract:(+"automated assessment"+"computer science 1") OR keywords.author.keyword:(+"automated assessment"+"computer science 1")

acmdlTitle:(+"automated assessment"+"CS1") OR recordAbstract:(+"automated assessment"+"CS1") OR keywords.author.keyword:(+"automated assessment"+"CS1")

acmdlTitle:(+"automatic assessment"+"introductory programming") OR recordAbstract:(+"automatic assessment"+"introductory programming") OR keywords.author.keyword:(+"automatic assessment"+"introductory programming")

acmdlTitle:(+"automatic assessment"+"introduction to programming") OR recordAbstract:(+"automatic assessment"+"introduction to programming") OR keywords.author.keyword:(+"automatic assessment"+"introduction to programming")

acmdlTitle:(+"automatic assessment"+"computer science 1") OR recordAbstract:(+"automatic assessment"+"computer science 1") OR keywords.author.keyword:(+"automatic assessment"+"computer science 1")
acmdlTitle:(+"automatic assessment"+"CS1") OR recordAbstract:(+"automatic assessment"+"CS1") OR keywords.author.keyword:(+"automatic assessment"+"CS1")
acmdlTitle:(+"juiz online"+"programação introdutória") OR recordAbstract:(+"juiz online"+"programação introdutória") OR keywords.author.keyword:(+"juiz online"+"programação introdutória")
acmdlTitle:(+"juiz online"+"introdução a programação") OR recordAbstract:(+"juiz online"+"introdução a programação") OR keywords.author.keyword:(+"juiz online"+"introdução a programação")
acmdlTitle:(+"juiz online"+"ciência da computação 1") OR recordAbstract:(+"juiz online"+"ciência da computação 1") OR keywords.author.keyword:(+"juiz online"+"ciência da computação 1")
acmdlTitle:(+"juiz online"+"CS1") OR recordAbstract:(+"juiz online"+"CS1") OR keywords.author.keyword:(+"juiz online"+"CS1")
acmdlTitle:(+"avaliação automática"+"programação introdutória") OR recordAbstract:(+"avaliação automática"+"programação introdutória") OR keywords.author.keyword:(+"avaliação automática"+"programação introdutória")
acmdlTitle:(+"avaliação automática"+"introdução a programação") OR recordAbstract:(+"avaliação automática"+"introdução a programação") OR keywords.author.keyword:(+"avaliação automática"+"introdução a programação")
acmdlTitle:(+"avaliação automática"+"ciência da computação 1") OR recordAbstract:(+"avaliação automática"+"ciência da computação 1") OR keywords.author.keyword:(+"avaliação automática"+"ciência da computação 1")
acmdlTitle:(+"avaliação automática"+"CS1") OR recordAbstract:(+"avaliação automática"+"CS1") OR keywords.author.keyword:(+"avaliação automática"+"CS1")
acmdlTitle:(+"correção automática"+"programação introdutória") OR recordAbstract:(+"correção automática"+"programação introdutória") OR keywords.author.keyword:(+"correção automática"+"programação introdutória")
acmdlTitle:(+"correção automática"+"introdução a programação") OR recordAbstract:(+"correção automática"+"introdução a programação") OR keywords.author.keyword:(+"correção automática"+"introdução a programação")
acmdlTitle:(+"correção automática"+"ciência da computação 1") OR recordAbstract:(+"correção automática"+"ciência da computação 1") OR keywords.author.keyword:(+"correção automática"+"ciência da computação 1")
acmdlTitle:(+"correção automática"+"CS1") OR recordAbstract:(+"correção auto-

mática"+"CS1") OR keywords.author.keyword:(+"correção automática"+"CS1")

APÊNDICE B

Protocolo da Revisão Sistemática da Literatura sobre Plágio e CS1

- **Questão principal**
  Qual é a estratégia mais adequada para identificar plágio em códigos-fonte de atividades de CS1?

- **Questões de pesquisa**
  -QP1: Quais estratégias já foram usadas para identificar plágio em códigos-fonte de atividades de CS1?
  - QP2: Como validar a estratégia usada para identificar plágio em códigos-fonte de atividades de CS1?

- **População**
  Artigos científicos que envolvem plágio e CS1.

- **Critério de seleção das bases de pesquisa**
  -Disponibilizar mecanismo de consulta via web.
  -Serem relacionadas a temas de Computação e Informática.
  -Permitir filtro por ano de publicação.

- **Bases de pesquisa selecionadas**
  - IEEEXplorer Digital Library
  - ACM Digital Library
  - Springer
  - Science Direct
  - Scopus

- **Idiomas**
  Português e Inglês

- **Strings de busca**
  ("plagiarism") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")

  ("plágio") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

- **Filtro da busca**
  Publicações entre 2010 a 2016. Pesquisar a *string* nos campos Título e Resumo.

- **Critérios para exclusão**
  - E1: Texto completo não disponível para acesso gratuitamente na web ou no portal de periódicos da Capes.
  - E2: Escrito outro idioma.
  - E3: Trabalho não aborda plágio e CS1.
  - E4: Não é um capítulo de livro com resumo ou artigo de periódico ou de conferência
  - E5: Publicação duplicada.

- **Critérios para inclusão**
  - I1: Aborda estratégias usadas para identificar plágio em códigos-fonte de atividades de CS1.
  - I2: Aborda validação de estratégias usadas para identificar plágio em códigos-fonte de atividades de CS1.

- **Fase de seleção**
  Os critérios de inclusão e exclusão serão aplicados considerando o título, palavras-chave e resumo.

- **Fase de extração**
  Os critérios de inclusão e exclusão serão aplicados considerando o texto completo.

- **Strings adaptadas à cada base**
  IEEEXplorer Digital Library
  ("plagiarism") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
  ("plágio") AND ("programação introdutória"OR "introdução a programação"OR

"ciência da computação 1"OR "CS1")

Springer
("plagiarism") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
("plágio") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

Science Direct
pub-date > 2009 and pub-date < 2017 and TITLE-ABSTR-KEY("plagiarism") AND TITLE-ABSTR-KEY("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
pub-date > 2009 and pub-date < 2017 and TITLE-ABSTR-KEY("plágio") AND TITLE-ABSTR-KEY("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

Scopus
TITLE-ABS-KEY (("plagiarism") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")) AND PUBYEAR > 2009 AND PUBYEAR < 2017
TITLE-ABS-KEY (("plágio") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")) AND PUBYEAR > 2009 AND PUBYEAR < 2017

ACM Digital Library
acmdlTitle:(+ "plagiarism"+ "introductory programming") OR recordAbstract:(+ "plagiarism"+ "introductory programming") OR keywords.author.keyword:(+ "plagiarism"+ "introductory programming")
acmdlTitle:(+ "plagiarism"+ "introduction to programming") OR recordAbstract:(+ "plagiarism"+ "introduction to programming") OR keywords.author.keyword:(+ "plagiarism"+ "introduction to programming")
acmdlTitle:(+ "plagiarism"+ "computer science 1") OR recordAbstract:(+ "plagiarism"+ "computer science 1") OR keywords.author.keyword:(+ "plagiarism"+ "computer science 1")
acmdlTitle:(+ "plagiarism"+ "CS1") OR recordAbstract:(+ "plagiarism"+ "CS1") OR keywords.author.keyword:(+ "plagiarism"+ "CS1")
acmdlTitle:(+ "plágio"+ "programação introdutória") OR recordAbstract:(+ "plágio"+ "programação introdutória") OR keywords.author.keyword:(+ "plágio"+

"programação introdutória")
acmdlTitle:(+ "plágio"+ "introdução a programação") OR recordAbstract:(+ "plágio"+ "introdução a programação") OR keywords.author.keyword:(+ "plágio"+ "introdução a programação")
acmdlTitle:(+ "plágio"+ "ciência da computação 1") OR recordAbstract:(+ "plágio"+ "ciência da computação 1") OR keywords.author.keyword:(+ "plágio"+ "ciência da computação 1")
acmdlTitle:(+ "plágio"+ "CS1") OR recordAbstract:(+ "plágio"+ "CS1") OR keywords.author.keyword:(+ "plágio"+ "CS1")

APÊNDICE C

Protocolo da Revisão Sistemática da Literatura sobre Dificuldade de Problemas de CS1

- **Questão principal**
  Qual é a estratégia mais adequada para medir a dificuldade de problemas de CS1?

- **Questões de pesquisa**
  -QP1: Quais estratégias já foram usadas para medir a dificuldade de problemas de CS1?
  - QP2: Como validar a estratégia para medir a dificuldade de problemas de CS1?

- **População**
  Artigos científicos que envolvem Dificuldade de Problemas e CS1.

- **Critério de seleção das bases de pesquisa**
  -Disponibilizar mecanismo de consulta via web.
  -Serem relacionadas a temas de Computação e Informática.
  -Permitir filtro por ano de publicação.

- **Bases de pesquisa selecionadas**
  - IEEEXplorer Digital Library
  - ACM Digital Library
  - Springer
  - Science Direct
  - Scopus

- **Idiomas**
  Português e Inglês

- **Strings de busca**

("difficulty") AND ("problem"OR "problems") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")

("dificuldade") AND ("problema"OR "problemas") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

- **Filtro da busca**
  Publicações entre 2010 a 2016. Pesquisar a *string* nos campos Título e Resumo.

- **Critérios para exclusão**
  - E1: Texto completo não disponível para acesso gratuitamente na web ou no portal de periódicos da Capes.
  - E2: Escrito outro idioma.
  - E3: Trabalho não aborda dificuldade de problemas e CS1.
  - E4: Não é um capítulo de livro com resumo ou artigo de periódico ou de conferência
  - E5: Publicação duplicada.

- **Critérios para inclusão**
  - I1: Aborda estratégias usadas para medir dificuldade de problemas de CS1.
  - I2: Aborda validação de estratégias usadas para medir dificuldade de problemas de CS1.

- **Fase de seleção**
  Os critérios de inclusão e exclusão serão aplicados considerando o título, palavras-chave e resumo.

- **Fase de extração**
  Os critérios de inclusão e exclusão serão aplicados considerando o texto completo.

- **Strings adaptadas à cada base**
  IEEEXplorer Digital Library
  ("difficulty") AND ("problem"OR "problems") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
  ("dificuldade") AND ("problema"OR "problemas") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

  Springer

("difficulty") AND ("problem"OR "problems") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
("dificuldade") AND ("problema"OR "problemas") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

Science Direct
pub-date > 2009 and pub-date < 2017 and TITLE-ABSTR-KEY("difficulty") AND TITLE-ABSTR-KEY("problem"OR "problems") AND TITLE-ABSTR-KEY("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")
pub-date > 2009 and pub-date < 2017 and TITLE-ABSTR-KEY("dificuldade") AND TITLE-ABSTR-KEY("problema"OR "problemas") AND TITLE-ABSTR-KEY("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")

Scopus
TITLE-ABS-KEY (("difficulty") AND ("problem"OR "problems") AND ("introductory programming"OR "introduction to programming"OR "computer science 1"OR "CS1")) AND PUBYEAR > 2009 AND PUBYEAR < 2017
TITLE-ABS-KEY (("dificuldade") AND ("problema"OR "problemas") AND ("programação introdutória"OR "introdução a programação"OR "ciência da computação 1"OR "CS1")) AND PUBYEAR > 2009 AND PUBYEAR < 2017

ACM Digital Library
acmdlTitle:(+ "difficulty"+ "problem"+ "introductory programming") OR recordAbstract:(+ "difficulty"+ "problem"+ "introductory programming") OR keywords.author.keyword:(+ "difficulty"+ "problem"+ "introductory programming")
acmdlTitle:(+ "difficulty"+ "problem"+ "introduction to programming") OR recordAbstract:(+ "difficulty"+ "problem"+ "introduction to programming") OR keywords.author.keyword:(+ "difficulty"+ "problem"+ "introduction to programming")
acmdlTitle:(+ "difficulty"+ "problem"+ "computer science 1") OR recordAbstract:(+ "difficulty"+ "problem"+ "computer science 1") OR keywords.author.keyword:(+ "difficulty"+ "problem"+ "computer science 1")
acmdlTitle:(+ "difficulty"+ "problem"+ "CS1") OR recordAbstract:(+ "difficulty"+ "problem"+ "CS1") OR keywords.author.keyword:(+ "difficulty"+ "problem"+ "CS1")

acmdlTitle:(+ "difficulty"+ "problems"+ "introductory programming") OR recordAbstract:(+ "difficulty"+ "problems"+ "introductory programming") OR keywords.author.keyword:(+ "difficulty"+ "problems"+ "introductory programming")
acmdlTitle:(+ "difficulty"+ "problems"+ "introduction to programming") OR recordAbstract:(+ "difficulty"+ "problems"+ "introduction to programming") OR keywords.author.keyword:(+ "difficulty"+ "problems"+ "introduction to programming")
acmdlTitle:(+ "difficulty"+ "problems"+ "computer science 1") OR recordAbstract:(+ "difficulty"+ "problems"+ "computer science 1") OR keywords.author.keyword:(+ "difficulty"+ "problems"+ "computer science 1")
acmdlTitle:(+ "difficulty"+ "problems"+ "CS1") OR recordAbstract:(+ "difficulty"+ "problems"+ "CS1") OR keywords.author.keyword:(+ "difficulty"+ "problems"+ "CS1")
acmdlTitle:(+ "dificuldade"+ "problema"+ "programação introdutória") OR recordAbstract:(+ "dificuldade"+ "problema"+ "programação introdutória") OR keywords.author.keyword:(+ "dificuldade"+ "problema"+ "programação introdutória")
acmdlTitle:(+ "dificuldade"+ "problema"+ "introdução a programação") OR recordAbstract:(+ "dificuldade"+ "problema"+ "introdução a programação") OR keywords.author.keyword:(+ "dificuldade"+ "problema"+ "introdução a programação")
acmdlTitle:(+ "dificuldade"+ "problema"+ "ciência da computação 1") OR recordAbstract:(+ "dificuldade"+ "problema"+ "ciência da computação 1") OR keywords.author.keyword:(+ "dificuldade"+ "problema"+ "ciência da computação 1")
acmdlTitle:(+ "dificuldade"+ "problema"+ "CS1") OR recordAbstract:(+ "dificuldade"+ "problema"+ "CS1") OR keywords.author.keyword:(+ "dificuldade"+ "problema"+ "CS1")
acmdlTitle:(+ "dificuldade"+ "problemas"+ "programação introdutória") OR recordAbstract:(+ "dificuldade"+ "problemas"+ "programação introdutória") OR keywords.author.keyword:(+ "dificuldade"+ "problemas"+ "programação introdutória")
acmdlTitle:(+ "dificuldade"+ "problemas"+ "introdução a programação") OR recordAbstract:(+ "dificuldade"+ "problemas"+ "introdução a programação") OR keywords.author.keyword:(+ "dificuldade"+ "problemas"+ "introdução a programação")
acmdlTitle:(+ "dificuldade"+ "problemas"+ "ciência da computação 1") OR re-

cordAbstract:(+ "dificuldade"+ "problemas"+ "ciência da computação 1") OR keywords.author.keyword:(+ "dificuldade"+ "problemas"+ "ciência da computação 1")
acmdlTitle:(+ "dificuldade"+ "problemas"+ "CS1") OR recordAbstract:(+ "dificuldade"+ "problemas"+ "CS1") OR keywords.author.keyword:(+ "dificuldade"+ "problemas"+ "CS1")

APÊNDICE D

Problema Soma

- **Descrição**
  O problema consiste em somar os valores de entrada agrupados em duplas.

- **Entrada**
  A entrada possui uma quantia par de números.
  A leitura da entrada termina quando os elementos do par da entrada tiverem valores iguais a 0 (zero).
- **Saída:**
  Para cada dois números somados será produzida uma linha contendo o valor da soma.
  A saída deverá ser escrita na saída padrão.

- **Exemplo de Entrada:**
  4
  7
  3
  10
  8
  7
  1001
  70
  0
  0
- **Exemplo de Saída:**
  11
  13
  15
  1071