ThinkCode: Um Ambiente Web para Apoio a Aprendizagem de Algoritmos Baseado em Conceitos de Complexidade Ciclomática

Denilson Rodrigues da Silva, Cristina Paludo Santos, Karla dos Santos Lencina

Universidade Regional Integrada do Alto Uruguai e das Missões

deniro@san.uri.br, paludo@san.uri.br, karla.krs@outlook.com

Abstract. *The teaching-learning process of algorithms and computer programming presents major challenges, from the understanding of the importance of the concepts worked and their purposes, to the motivation of the learner through innovative methodologies and subsidies capable of awakening it. In this sense, the present article presents the development of a Web environment, supported by the principles of meaningful learning and reflexive learning, to aid in the learning of algorithms through the analysis of the efficiency of algorithmic solutions. Finally, it presents the results of the application of the platform in programming classes of a higher education institution.*

Resumo. *O processo de ensino-aprendizagem de algoritmos e programação de computadores apresenta grandes desafios, desde o entendimento da importância dos conceitos trabalhados e suas finalidades, até a motivação do aprendiz através de metodologias e subsídios inovadores capazes de despertá-la. Nesse sentido, o presente artigo apresenta o desenvolvimento de um ambiente Web, sustentado pelos princípios de aprendizagem significativa e aprendizagem reflexiva, para auxiliar a aprendizagem de algoritmos através da análise de eficiência de soluções algorítmicas. Por fim, apresenta os resultados da aplicação da plataforma em turmas de programação de uma Instituição de Ensino Superior.*

1. Introdução

A Computação está presente em todas as atividades humanas, das artes às tecnologias (Nunes,2008), o que torna indispensável que profissionais dessa área sejam capazes de aplicar conceitos trabalhados em sua formação da melhor forma possível, trazendo soluções eficientes para problemas enfrentados, seja no seu cotidiano ou em âmbito profissional. Assim, faz-se necessária a existência de profissionais de senso crítico e inovador, que sejam capazes de unir inteligência e dinamismo na busca por melhores soluções.

Essas habilidades se referem ao pensamento computacional (PC) que segundo Wing (2014) é uma das principais contribuições da ciência da computação para o mundo, "pois permite que os profissionais pensem nos problemas de forma analítica e desenvolvam soluções computacionais para os mais diversos domínios de aplicação" (Ajala et al., 2016). Para despertar essa capacidade desde os primórdios da aprendizagem é necessário orientar o aluno a preocupar-se com o desenvolvimento de códigos eficientes, não somente funcionais. Preocupar-se com a eficiência do código, caracteriza o pensamento algorítmico, que está vinculado ao pensamento computacional, como sendo a etapa de reavaliação da solução criada para um determinado problema, citada por Wing (2006) ao apresentar o pensamento computacional pela primeira vez.

O ensino de programação, fortemente vinculado ao ensino de Algoritmos e Estruturas de Dados, baseia-se nesses princípios e busca "capacitar o aluno a desenvolver algoritmos

DOI: 10.5753/sbie.2022.225727

computacionais através do fornecimento dos fundamentos básicos para programação de computadores" (Júnior, 2016). Assim, a disciplina de Algoritmos é fundamental e espera-se que os alunos desenvolvam um conjunto de habilidades cognitivas para que possam efetivamente aprender a resolver problemas utilizando o computador como ferramenta (Giraffa et al., 2003).

Porém, a didática tradicional de ensino – que reside na explanação de definições e exemplos, seguida da resolução de exercícios com aumento gradativo de níveis de dificuldade – não desperta o interesse do aluno, pois não fica claro em seu primeiro ano de curso a importância de certos conteúdos e qual a relação destes com os demais conceitos trabalhados durante sua formação (Júnior, 2016). Assim, o presente trabalho, busca suprir lacunas existentes no processo de ensino e de aprendizagem de Algoritmos e Estruturas de Dados por meio de um ambiente Web baseado no modelo computacional descritivo que utiliza da análise da Complexidade Ciclomática e de noções de Complexidade Espacial para auxiliar nas fases iniciais de aprendizagem algorítmica, bem como para aproximar os conceitos de pensamento computacional da realidade de acadêmicos dos cursos de computação.

O ambiente fornece todos os recursos necessários para credenciamento de estudantes, além de persistência das análises realizadas em códigos inseridos por esses alunos na plataforma, tornando possível a visualização do progresso desse aluno perante suas análises, bem como um sistema de ranqueamento para tornar o processo mais estimulante, trabalhando com o princípio de competitividade de forma a estimular o aluno a desenvolver soluções melhores. Desta forma, o propósito é despertar o que Schön (2000) denominou reflexão-na-ação, quando o aluno passa do "conhecer-na-ação" – saber fazer um algoritmo, conhecer seu processo de execução e desenvolvimento – para uma reflexão mais profunda sobre quais práticas serão mais eficientes, estimulando o pensamento algorítmico desde as fases iniciais de aprendizagem, o que poderá ser um ponto decisivo para efetivar um bom profissional ao final da graduação (Silva, 2020).

Uma descrição mais detalhada do ambiente proposto é apresentada nas seções subsequentes. A seção 2 apresenta uma visão geral das teorias de aprendizagem que nortearam o processo de desenvolvimento do ambiente, bem como alguns trabalhos relacionados, enfatizando o modelo descritivo que fundamenta o desenvolvimento do ThinkCode; a seção 3 descreve as adaptações realizadas para contemplar a ampliação das funcionalidades propostas e as principais interfaces do ambiente; a seção 4 descreve o processo de validação realizado e os resultados obtidos. Por fim, a seção 5 apresenta as considerações finais e direcionamentos futuros.

2. Princípios Norteadores

O desenvolvimento do ambiente proposto fundamenta-se em duas teorias de aprendizagem que compreendem: a aprendizagem significativa de David Ausubel e a aprendizagem reflexiva de Donald Schön, por possuírem como foco o aprendizado através da participação ativa do estudante. A primeira define a aprendizagem através do relacionamento de novas ideias com o conhecimento prévio do estudante, implicando que cada conhecimento adquirido modifica a estrutura cognitiva do indivíduo (Moreira, 2009). A segunda preconiza a aprendizagem através da prática, motivando o aluno a refletir diante de suas ações, proporcionando a reavaliação e reflexão (Schön, 2000). Assim, o ambiente busca despertar no estudante a capacidade de desenvolver soluções algorítmicas mais eficientes pela análise de soluções, introduzindo ideias de eficiência e qualidade nos conhecimentos que já possui previamente sobre desenvolvimento de algoritmos – processo este sedimentado pela aprendizagem significativa – estimulando, após

os resultados da análise do algoritmo apresentado, a reflexão dos alunos durante o desenvolvimento de novas soluções – aprendizagem reflexiva.

Essas teorias empregam-se no desenvolvimento do ambiente ThinkCode como complementares e colaborativas, com propósito de auxiliar no processo de constituição do pensamento algorítmico e consequentemente o pensamento computacional.

2.1. Ambientes de Apoio à Aprendizagem de Algoritmos

Ambientes virtuais de apoio à aprendizagem funcionam como um conjunto de elementos tecnológicos para facilitar o processo de aprendizagem, rompendo os limites de uma sala de aula presencial e podem ajudar os alunos a se desenvolverem mais rápido (Silva et al., 2020).

Há diversas propostas de ferramentas no âmbito do ensino de algoritmos e cada trabalho apresenta um foco relacionado às necessidades específicas. Amaral et al. (2017) desenvolveram uma plataforma Web baseada em conceitos e teorias de aprendizagem, no qual o estudante pode realizar a construção do conhecimento sobre programação, de forma autônoma e à distância. Ao colocarem o ambiente Algo+ em prática constataram através da análise de um instrumento de pesquisa que houve desempenho satisfatório dos alunos que o utilizaram, apontando a efetividade na adoção do Algo+ no processo de aprendizagem.

Uma ferramenta já consolidada de apoio ao ensino e aprendizagem de algoritmos é a URI Online Judge, apresentada por Tonin e Bez (2013). Esse portal contém problemas no estilo do ICPC (*International Collegiate Programming Contest*) da ACM e fornece ao usuário um juiz online para testar. É um projeto em constante desenvolvimento e tornou-se o ambiente mais completo e robusto na área, possuindo inúmeros componentes e ferramentas integrados.

Veras et al. (2010) apresentaram o SEED (Software de Ensino de Estruturas de Dados) para auxiliar a aprendizagem de algoritmos de ordenação de dados. A ferramenta foi desenvolvida em Java e é dividida nos módulos de aprendizagem e de comparação. No primeiro o usuário seleciona um método disponível e visualiza seu princípio, pseudocódigo e sua análise da complexidade. No segundo o usuário seleciona métodos disponíveis e avalia o desempenho de dois ou mais métodos de ordenação através de um gráfico que relaciona o tempo ou a quantidade de comparações com a quantidade de elementos a ser ordenado, tendo assim uma visão do comportamento assintótico dos algoritmos.

Todos os trabalhos até aqui citados contêm importantes contribuições para auxiliar no ensino das disciplinas de algoritmos e programação, porém nenhum aborda aspectos vinculados à reflexão no desenvolvimento de soluções em relação a eficiência de códigos, o pensamento algorítmico propriamente dito. O último se assemelha por trabalhar noções de complexidade em um ramo específico de estrutura de dados, e difere-se desta proposta por inclui um público em processo inicial de aprendizagem de programação.

No presente trabalho destaca-se a proposta de Ajala e Silva (2016) que desenvolveram o modelo descritivo que dá sustentação ao desenvolvimento do Ambiente ThinkCode. O modelo se utiliza da Complexidade Ciclomática como principal indicador de eficiência de código e descreve o processo da análise de soluções algorítmicas de um determinado problema. Nessa proposta há uma base de conhecimento que armazena enunciados de problemas e para cada um deles uma solução de referência. Assim, a solução desenvolvida pelo aluno é comparada com a solução de referência no processo da análise.

O cálculo da complexidade ciclomática realizado na análise mensura códigos através das estruturas de controle de fluxo das soluções, contabilizando o número de caminhos independentes do conjunto base principal do programa através de um grafo de fluxo que detalha cada instrução e fluxo contidos no código, e retorna o número mínimo de testes para satisfazer

todos esses caminhos. O valor retornado pelo cálculo no modelo é analisado quanto a ser baixo ou alto em comparação ao resultado da complexidade do código referencial, definindo a eficiência da solução desenvolvida pelo aluno.

A cada submissão uma nova análise da solução referencial e do aluno será realizada. Isso ocorre porque a base não possui suporte para o armazenamento dos resultados obtidos, sendo necessário o reprocessamento nos casos de submissões. A Linguagem C é a única linguagem suportada pelo modelo, sendo que os comandos *case* e *do while* foram ignorados nessa proposta, por possuírem peculiaridades que dificultam o tratamento, sendo que as estruturas de armazenamento dos comandos não são adequadas para suportar esses comandos.

3. O Ambiente ThinkCode

3.1.Adaptações do Modelo Descritivo de Ajala e Silva

O Ambiente ThinkCode abrange aspectos não considerados por Ajala e Silva (2016). O novo modelo possui duas etapas definidas e distintas, a coleta de dados e a análise de dados, sendo que a segunda depende dos dados obtidos pela primeira. Outro aspecto considerado é a generalidade das análises e para contemplá-lo foi necessária a inclusão de persistência dos dados das estruturas das linguagens de programação, bem como dos resultados das análises dos códigos fontes, para evitar retrabalho e reprocessamento na comparação de soluções.

A Complexidade Ciclomática sozinha não permite definir a eficiência real do código-fonte porque trata apenas da avaliação das estruturas de fluxo do código, além disso o valor gerado por essa métrica não pode indicar a eficiência total de uma solução. Assim, para tornar a análise mais precisa foi necessário incluir outros indicadores que retiram dados quantitativos de variáveis e operadores lógicos e linhas úteis do código e passam a fazer parte, juntamente com a complexidade ciclomática de uma média de análise da solução, que indicará a eficiência da solução através de todos esses dados.

Outra atualização do modelo inclui a alteração do princípio de solução referencial pelo ranqueamento de soluções através dos resultados obtidos nas análises, sendo que as soluções consideradas melhores serão as que possuírem os menores indicadores, por utilizarem menos recursos para resolver o problema. Esse aspecto torna a análise mais interessante porque parte-se da ideia de que a submissão de uma nova solução para análise pode substituir a posição da anterior e passar a ser a considerada referencial por estar nas primeiras colocações do ranking.

A base de conhecimento é um elemento crucial do modelo. Diferentemente da proposta anterior, é através da base de conhecimento que são fornecidos os dados sustentadores da análise. Não só são armazenadas definições de algoritmos a serem solucionados, como também toda a estrutura das linguagens de programação suportadas e os resultados obtidos do processamento das submissões. Para que isso fosse possível os dados armazenados foram organizados de forma lógica através de um banco de dados relacional.

As estruturas de desvio são componentes indispensáveis para a geração do grafo da complexidade ciclomática. Por esta razão, Ajala e Silva (2016) afirmam que “é necessário identificar no código-fonte as estruturas responsáveis por criar desvios de fluxo na execução do código”. Existem dois tipos de estruturas de controle que causam desvios de fluxo em um código: as estruturas condicionais e as estruturas de repetição. As primeiras funcionam como pontos de decisão e são os casos das estruturas *if, else* e *case* na Linguagem C e *se, senão* e *caso* no Portugol. As últimas estruturas, diferem das primeiras por envolverem a execução de um determinado trecho repetidamente dependendo também de uma condição e são representadas pelos comandos *for, while* e *do while* da Linguagem C e no Portugol por *para, enquanto e repete*. A Tabela 1 apresenta o comparativo entre os dois modelos.

Tabela 1. Comparativo entre os modelos descritivos

Modelo Descritivo de Ajala e Silva (2016)	Modelo Adaptado
Sem distinção de etapas	Duas etapas definidas: ● Coleta de dados ● Análise de Dados
Base de conhecimento armazena: ● Descrições de problemas ● Soluções referenciais da literatura	Base de conhecimento armazena: ● Dados das linguagens suportadas ● Descrições de problemas ● Soluções submetidas ● Resultados das análises
Sistema de arquivos	Base de dados relacional
Indicadores de análise: ● Complexidade Ciclomática ● Número de regiões do grafo ● Número de arestas do grafo ● Número de vértices ● Número de comandos de desvio	Indicadores de análise: ● Complexidade Ciclomática ● Número de regiões do grafo ● Número de arestas do grafo ● Número de vértices ● Número de comandos de desvio ● Número de variáveis declaradas ● Número de conectivos lógicos ● Número de linhas úteis ● Média da solução
Complexidade Ciclomática como principal indicador da análise (impreciso em determinados casos)	Média da solução baseada na Complexidade Ciclomática e demais indicadores (maior precisão)
Sem tratamento para os comandos *case* e *do while*	Tratamento dos comandos *case* e *do while*
Suporte a uma linguagem de programação: Linguagem C	Suporte a qualquer linguagem de programação que possua delimitadores de comandos: como chaves ({}).
Solução referencial x Solução submetida	Comparação entre soluções submetidas e salvas
Sem rankeamento de soluções	Com rankeamento de soluções
Estruturas menos definidas	Estruturas mais definidas

A notação de grafo representa graficamente os comandos de desvio e dá uma visão geral da estrutura do código-fonte. Cada comando possui um representante nessa notação, mostrado na Figura 1.

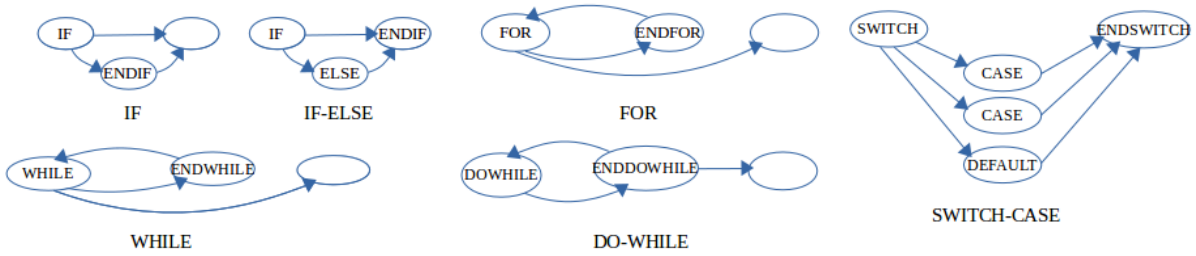

Figura 1. Notação dos comandos de desvio no grafo

Cada comando é seguido de seu terminal de comando representado pelo vértice END ou FIM e a sequência do fluxo é representada pelas arestas direcionadas. O valor que define a posição de um código é dado pela média ponderada dos valores dos indicadores complexidade ciclomática, número de comandos de desvio, número de regiões do grafo de fluxo, número de variáveis, número de linhas úteis e número de conectivos lógicos retirados na análise e descritos anteriormente.

As quantidades de arestas e vértices do grafo não constam na média porque fazem parte do cálculo da Complexidade Ciclomática, e ao incluí-los seriam contabilizados duas vezes. Cada indicador incluído na média detém um peso de acordo com a sua importância na avaliação. Os pesos maiores definem uma maior relevância para o indicador dentro da média da solução. Essa distribuição de valores é apresentada na Tabela 2.

Tabela 2. Distribuição dos pesos para o cálculo da média da solução

Indicadores	**Peso**
Complexidade Ciclomática.	Seis (6)
Quantidade de comandos de desvio encontrados no código-fonte.	Cinco (5)
Número de regiões do grafo.	Quatro (4)
Quantidade de variáveis encontradas no código-fonte.	Três (3)
Número de conectivos lógicos encontrados no código-fonte.	Dois (2)
Número de linhas úteis do código-fonte.	Um (1)

O peso de maior importância continua sendo a Complexidade Ciclomática, seguida de indicadores que possuem relações mais próximas a essa métrica. Outro ponto relevante para a distribuição dos pesos foi em relação ao objetivo que se deseja alcançar com este trabalho. Nossas preocupações são centradas em instigar no aluno uma reflexão mais profunda diante do processo de desenvolvimento em relação ao fluxo do código e como poderia utilizar menos recursos da linguagem para chegar à solução de um problema. Os pesos foram organizados sequencialmente em ordem decrescente e suas definições se dão pela proximidade do indicador com a Complexidade Ciclomática. Assim, a Complexidade Ciclomática detém o maior peso e as demais recebem a ordenação conforme suas relações com a métrica.

3.2. Principais Interfaces do Ambiente

A interface inicial do ambiente, representada na Figura 2, fornece ao usuário uma breve explicação das funcionalidades do ThinkCode. Os dados do usuário com sessão ativa são encontrados no topo superior esquerdo, logo acima do menu de navegação (*omitindo na imagem em função da blind review*). Nesse menu a opção *Home* encontra-se selecionada e logo abaixo a listagem de problemas e as configurações de conta do usuário.

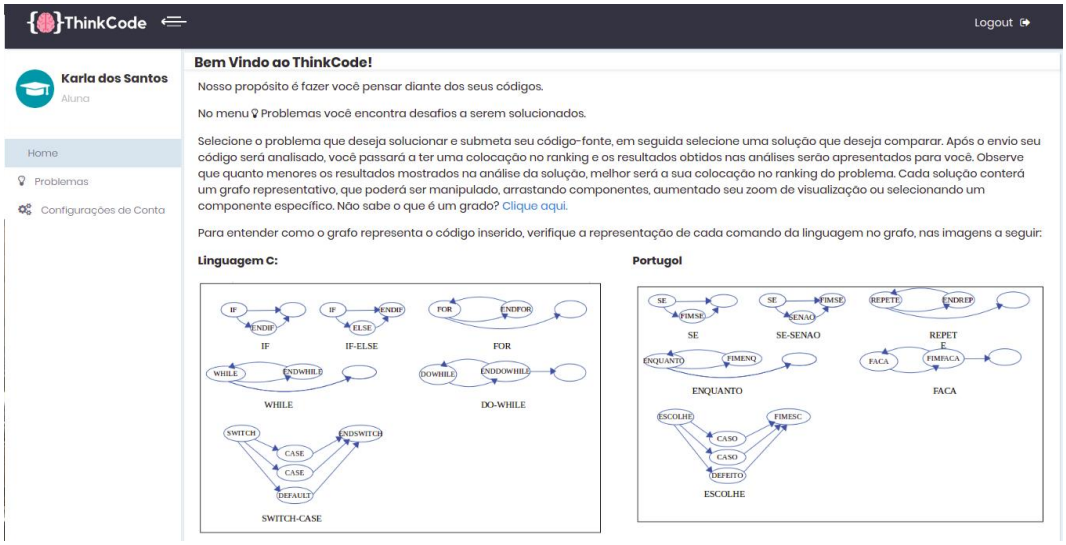

Figura 2. Interface principal do Ambiente ThinkCode

O usuário deve selecionar um problema algorítmico da listagem e ao clicar no botão *Enviar Solução*, contido na tela detalhamento do problema, ele será redirecionado para a tela mostrada na Figura 3, que ilustra a situação após o preenchimento dos dados necessários para realizar a análise. Para concluir o processo o usuário deve clicar no botão *Analisar Solução* que dispara a requisição de análise.

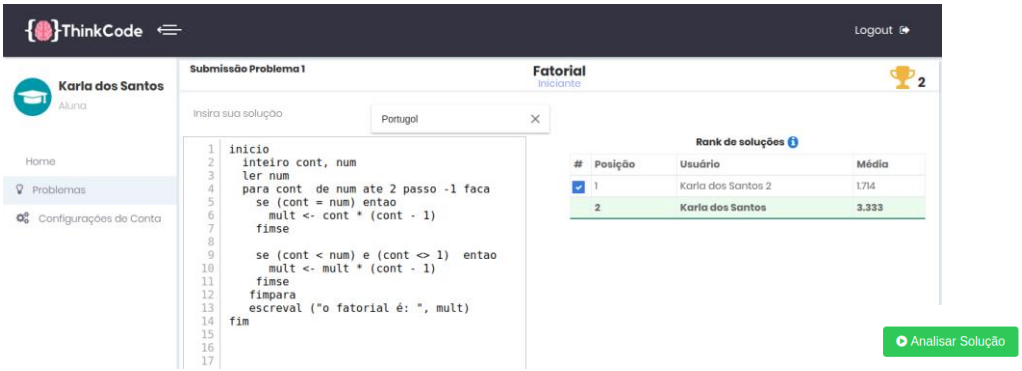

Figura 3. Interface de submissão

Ao término o usuário será direcionado para a interface de resultados, representada na Figura 4. No canto superior direito é mostrada a posição obtida pelo código do usuário no ranking do problema, neste caso o problema do *Fatorial*. Ao lado da colocação é possível visualizar o ranking completo de soluções clicando no botão em azul. Os grafos são apresentados no meio da tela, sendo o primeiro a representação do código submetido e o segundo a solução de um código já salvo no sistema, submetido por outro usuário.

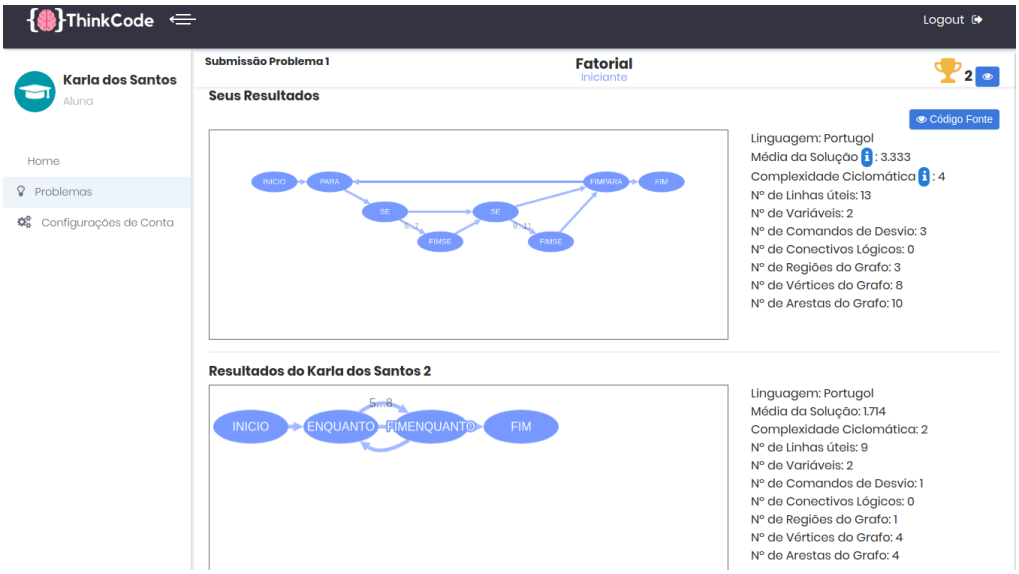

Figura 4. Interface do resultado da análise

Ao lado encontram-se os indicadores, juntamente com botões informativos que ao serem clicados mostram um texto explicativo do que os valores representam. O usuário pode visualizar o código-fonte inserido novamente através do botão *Código-fonte* localizado no canto superior direito, logo abaixo de sua colocação.

4. Validação do Ambiente e Resultados Obtidos

Com o intuito de compreender o impacto do ambiente e, consequentemente, do modelo em que está alicerçado, realizou-se um estudo de caso diante da utilização do ambiente ThinkCode por alunos do 1º e 2º ano do curso de Ciência da Computação da Universidade Regional Integrada do Alto Uruguai e das Missões (URI Santo Ângelo). Participaram do processo avaliativo 27 estudantes.

O *corpus* de análise dessa pesquisa foi obtido através das percepções dos alunos após a utilização, das observações dos pesquisadores diante das interações dos alunos com o ambiente

e das informações contidas na base de dados após a experiência. Para a coleta de dados utilizou-se um questionário constituído por onze questões, sendo oito objetivas e três subjetivas. Tratam-se de perguntas relacionadas tanto ao desenvolvimento de código quanto às informações dispostas pelo sistema. Cabe destacar que o caráter deste estudo é investigativo e tem a intenção de explorar as percepções tanto dos acadêmicos quanto do próprio pesquisador, por isso utiliza de uma perspectiva de ordem mais qualitativa, incluindo alguns elementos quantitativos, no processo de interpretação dos resultados.

A maioria dos discentes entrevistados respondeu já possuir uma preocupação em relação à qualidade no desenvolvimento algorítmico, sendo que grande parte deles destacou que em um primeiro momento preocupa-se em obter a solução correta para depois refiná-la. Isso indica que os alunos já possuíam características do pensamento algorítmico e que o ambiente servirá como um instrumento de apoio ao processo já trabalhado em aula pelos docentes da universidade. Essas características foram obtidas pela primeira pergunta do questionário, em que os estudantes deveriam dissertar sobre a sua maior preocupação ao resolver um algoritmo.

Todos os participantes consideraram as propostas de desafios apresentadas condizentes com suas habilidades de programação. Isso aponta que a preocupação existente na seleção dos desafios foi importante e que esses foram adequados aos níveis de aprendizado do público alvo.

Outra unanimidade foi em relação às diferenças encontradas nas comparações realizadas. Todos confirmaram que puderam notar diferenças entre as soluções que submeteram na plataforma com as que se encontravam cadastradas e selecionadas para a comparação, sendo que 66,7% (18 alunos) apontaram que essas diferenças puderam ser notadas tanto pelos grafos de fluxo apresentados, quanto pelos indicadores numéricos presentes. Ainda 18,5% (5 alunos) indicaram as pontuações como elemento mais significativo na análise, e 14,8% (4 alunos) apontaram o grafo como maior indicador de distinções. Esses dados indicam que as comparações realizadas pelo ambiente foram significativas, sendo reforçadas por uma maioria de 85,2% dos alunos que informaram que as distinções entre as soluções estavam claras diante da análise apresentada na ferramenta.

Quando indagados sobre quais as maiores diferenças notadas diante das análises, a maioria dos estudantes respondeu sobre as diferenças nas estruturas utilizadas. Alguns citaram diretamente a estrutura mostrada no grafo, outros mencionaram os indicadores de quantidade de linhas e/ou variáveis utilizadas. Alguns relacionaram ainda as informações apresentadas com desempenho e complexidade do algoritmo. Dois apontamentos chamaram a atenção que são:

> *"A quantidade em grafos, mas necessita uma explicação maior sobre a diferença de conter mais ou menos grafos."*
>
> *"A forma de fazer o algoritmo, e o grafo ajudaram a ver como poderia ter feito melhor no algoritmo."*

Estas opiniões contrapõem-se, enquanto uma afirma que o grafo foi um elemento importante na construção de sua reflexão e outra informa que as estruturas apresentadas não ficaram muito claras. Pela discordância ocorrida e também pelo público da pesquisa incluir estudantes em diferentes níveis de aprendizagem, é possível considerar aspectos a serem melhorados no ambiente.

Em relação às colocações obtidas frente às soluções comparadas 81,5% (22 alunos) responderam que ficaram, em algum momento, com uma colocação melhor que a selecionada para a comparação. Essa porcentagem não necessariamente indica que os 18,5% dos alunos obtiveram as últimas colocações do ranking. Isso porque existem mais dois cenários possíveis: o primeiro em que o aluno poderia ter selecionado as primeiras soluções do ranking para comparação e não as superou e, em segundo, em que selecionou as demais, sendo estas

relacionadas a códigos encontrados em colocações do ranking medianas. Sobre a significância dessas colocações diante de suas percepções muitos indicaram que representaram aspectos da qualidade e performance no desenvolvimento do código e discorreram sobre a possibilidade de melhorá-lo nesses níveis.

Esses aspectos reforçam que houve uma reflexão por parte do aluno diante dos dois cenários possíveis. Em um cenário de sucesso, em que a solução obteve uma colocação melhor, o aluno parte para a observação do que ela significa diante da outra e discorre sobre aspectos de eficiência em relação à outra, refletindo sobre as estruturas diferentes usadas no código-fonte. Frente a uma situação de insucesso, o aluno passa a pensar em quais aspectos poderia melhorar, tentando abstrair conceitos de estruturas através do grafo da solução adversária.

Houve também observações diante da opção de comparação de soluções do próprio aluno, onde cada solução submetida seria mantida, podendo ser selecionada uma solução anterior para comparar com a nova solução reformulada. Essa situação é tratada pelo modelo, pois não há nenhuma restrição de quantidade de soluções que pode ser submetida por usuário para um mesmo problema.

5. Considerações Finais

Este trabalho apresentou o desenvolvimento de um ambiente de aprendizagem, pautado na evolução de um modelo descritivo que utiliza da análise de soluções algorítmicas através do cálculo da Complexidade Ciclomática para auxiliar nos processos de constituição de pensamento algorítmico e computacional por alunos de computação. Para tanto o presente estudo reestruturou um modelo pré-definido, distinguindo e delimitando suas etapas de funcionamento, tornando-o mais generalista quanto à suas aplicações em diferentes Linguagens de Programação, atribuindo mapeamento e persistência de dados tanto para o armazenamento de informações essenciais ao seu funcionamento quanto para os resultados gerados através de sua análise, incluiu novos indicadores que reforçam as noções de eficiência trabalhadas e aumentam a precisão da análise, adicionou comandos das linguagens de programação que eram desprezados em sua primeira versão e incluiu dados da pseudolinguagem Portugol na avaliação.

Numa perspectiva empírica, foi possível investigar a compreensão de alunos diante das funcionalidades do ambiente e da abordagem proposta e através desse olhar e de outras observações levantadas por experiência prática, identificou aspectos positivos da ferramenta e do modelo discutidos ao longo deste trabalho. Neste sentido destacam-se como pontos positivos as aplicações reais dos conceitos de aprendizagem reflexiva e significativa, que foram supridos e trabalhados pelos recursos disponibilizados no ambiente e modelo. O armazenamento de diversos tipos de soluções submetidas como repositório válido para utilização na aprendizagem de algoritmos e programação, aliadas aos seus grafos de fluxo que ajudam na abstração e aproximam alunos iniciantes de conceitos trabalhados posteriormente no curso.

No âmbito deste trabalho, os entendimentos sobre os contornos do pensamento computacional estão ancorados nos conceitos-chaves de abstração, pensamento algorítmico, avaliação, generalização e decomposição (Wing, 2006, 2014; Selby, 2013; Silva 2020). Neste sentido, a partir do modelo descritivo apresentado, dos recursos e funcionalidades implementados e das experiências realizadas, entende-se que o ThinkCode oferece meios para potencializar o processo inicial de desenvolvimento do pensamento computacional em alunos ingressantes de cursos de computação. Objetivamente, o ThinkCode estimula de modo mais intenso três habilidades: (1) O pensamento algorítmico - que considera a capacidade de definir com precisão e clareza os passos necessários para resolver problemas; (2) Avaliação - que inclui

a capacidade de avaliar processos em termos de eficiência e uso de recursos, bem como a capacidade de reconhecer e avaliar resultados e, (3) Generalização - como um processo cognitivo que visa resolver novos problemas com base em soluções semelhantes já estabelecidas. Este conceito envolve a capacidade de verificar características funcionais e estruturais comuns a diferentes contextos e situações.

Em linhas gerais pôde-se observar que o ambiente desenvolvido apresentou evidências positivas e potencialidades para apoiar o processo de aprendizagem de conceitos e práticas de computação, sendo necessários esforços de continuidade de avaliações e aprimoramentos, a fim de colher resultados em longo prazo.

Referências Bibliográficas

Ajala, V.; Silva, D.; Paludo, C; e Rolim, C. Um Modelo para Análise da Complexidade Ciclomática no Processo Inicial de Constituição do Pensamento Algorítmico. *Anais do XXXVI Congresso da Sociedade Brasileira de Computação*, p. 587–596, 2016.

Amaral, Érico et al. Algo+ uma ferramenta para o apoio ao ensino de algoritmos e programaçao para alunos iniciantes. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. 2017. p. 1677.

Moreira, Marco Antonio; Masini, Elcie Aparecida Fortes Salzano. Aprendizagem significativa: a teoria de David Ausubel. São Paulo, SP: Centauro, 2009.

Giraffa, Lucia; Marczak, Sabrina; Almeida, Gláucio. O ensino de algoritmos e programação mediado por um ambiente na Web. In: **Congresso Nacional da Sociedade Brasileira de Computação (SBC'2003). Campinas, SP, Brasil**. sn, 2003.

Junior, H. de S. C., Prado, A. F. e Araújo, M. A. P. (2016). Complexity tool : uma ferramenta para medir complexidade ciclomática de métodos java. *Multiverso: Revista Eletrônica do campus Juiz de Fora*, v. 1, n. 2016, p. 66–76.

Rodrigues da Silva, Denilson; Kurtz, Fabiana Diniz; Paludo Santos, Cristina. Computational thinking and TPACK in science education: a southern-Brazil experience. **Paradigma**, v. 41, n. 2, 2020.

Schön, D. A. Educando o profissional reflexivo: um novo design para o ensino e a aprendizagem. Porto Alegre: ARTMED, 2000.

Selby, C. C.; Woollard, J. Computational Thinking : The Developing Definition. In **ITiCSE Conference 2013**. Canterbury, England: University of Southampton (E-prints), 2013.

Silva, Denilson Rodrigues. Desenvolvimento do pensamento computacional como dimensão estruturante da atividade do professor de cursos superiores de computação. Tese de doutorado-UNIJUÍ, 182 f., 2020.

Tonin, Neilor Avelino; Bez, Jean Luca. Uri online judge: A new interactive learning approach. **Computer Technology and Application**, v. 4, n. 1, 2013.

Veras, Rodrigo MS et al. Ferramenta computacional para o ensino de algoritmos de ordenação. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. 2010.

Wing, J. M. Computational thinking. **Communications of the ACM**, 49(3), 33–35, 2006.

Wing, J. M. Computational thinking benefits society. **40th anniversary blog of social issues in computing**, p. 26, 2014.