UNIVERSIDADE FEDERAL DE OURO PRETO

DEPARTAMENTO DE COMPUTAÇÃO

Palloma Stéphanne Silva Brito

O USO DE FERRAMENTAS COMPUTACIONAIS PARA O ENSINO DE PROGRAMAÇÃO PARA ALUNOS DE ENGENHARIA

Ouro Preto, MG
2019

Palloma Stéphanne Silva Brito

UNIVERSIDADE FEDERAL DE OURO PRETO

DEPARTAMENTO DE COMPUTAÇÃO

Monografia II apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. MSc. Reinaldo Silva Fortes

Ouro Preto, MG
2019

B777u Brito, Palloma Stephanne Silva.
O uso de ferramentas computacionais para o ensino de programação para alunos de engenharia [manuscrito] / Palloma Stephanne Silva Brito. - 2019.

38f.: il.: color; grafs; tabs.

Orientador: Prof. MSc. Reinaldo Silva Fortes.

Monografia (Graduação). Universidade Federal de Ouro Preto. Instituto de Ciências Exatas e Biológicas. Departamento de Computação.

1. Programação (Computadores). 2. Raciocínio . 3. Controle automático. I. Fortes, Reinaldo Silva. II. Universidade Federal de Ouro Preto. III. Titulo.

CDU: 004.42

Catalogação: ficha.sisbin@ufop.edu.br

Palloma Stéphanne Silva Brito

O USO DE FERRAMENTAS COMPUTACIONAIS PARA O ENSINO DE PROGRAMAÇÃO PARA ALUNOS DE ENGENHARIA

Monografia II apresentada ao Curso de Ciência da Computação da Universidade Federal de Ouro Preto como parte dos requisitos necessários para a obtenção do grau em Bacharel em Ciência da Computação.

Aprovada em Ouro Preto, 18 de Dezembro de 2019.

Prof. MSc. Reinaldo Silva Fortes
Universidade Federal de Ouro Preto
Orientador

Prof. Dr. Gustavo Peixoto Silva
Universidade Federal de Ouro Preto - UFOP
Examinador

Prof. Dr. Rodrigo Geraldo Ribeiro
Universidade Federal de Ouro Preto - UFOP
Examinador

Dedico este trabalho ao meu falecido pai, Wallace Jorge de Brito.

Agradecimentos

À Deus, pelo dom da vida e por permitir que eu chegasse até aqui.

Aos meu familiares, pelo intenso incentivo para que eu não desistisse. Em especial a minha mãe, Marilene e minha avó, Lindomar.

A todos os meus amigos, em especial ao Bruno, ao Marco e a Layla, os quais estiveram comigo durante toda minha graduação, sendo verdadeiros irmãos.

A todos os professores do DECOM e em especial ao meu orientador, Profº Reinaldo, por me conduzir tão bem durante todos os trabalhos desenvolvidos.

Aos membros do Projeto opCod3rs, por me proporcionar viver momentos ímpares no ensino da programação.

Por fim, agradeço a todos que de alguma forma contribuíram para que este momento chegasse.

Resumo

Disciplinas correlatas a Algoritmos e Programação devem estar presentes na grade curricular dos cursos de graduação de diversas Engenharias. Tais disciplinas são consideradas muito desafiadoras visto que exigem o desenvolvimento de estratégias de solução de problemas com base lógico-matemática. O alto nível de abstração exigido nessas disciplinas causa desmotivação nos alunos, fator preponderante para as altas taxas de reprovações e desistências. Diversas ferramentas vêm sendo exploradas no intuito de tornar mais dinâmico o processo ensino-aprendizagem e atenuar as dificuldades dos alunos em aprender o conteúdo abordado. Nesse contexto, o presente trabalho aborda uma metodologia de ensino que utiliza um corretor automático de códigos e exploração de questões objetivas como forma de consolidar os assuntos abordados durante as aulas teóricas, mantendo o aluno informado sobre seu desenvolvimento no curso. Essa metodologia foi aplicada e avaliada para algumas turmas dos cursos de Engenharia da Universidade Federal de Ouro Preto (UFOP) demostrando o potencial positivo no aprendizado dos alunos e, consequentemente, em seu desempenho acadêmico na disciplina.

Palavras-chave: Programação. Raciocínio Lógico. Corretor automático.

Abstract

Disciplines related to Algorithms and Programming must be present in the curriculum of undergraduate courses of several Engineering. Such disciplines are considered very challenging since they require the development of problem-solving strategies on a logical-mathematical basis. The high level of abstraction required in these disciplines causes students to become unmotivated, a preponderant factor in the high rates of disapprovals and dropouts. Several tools have been explored in order to make the teaching-learning process more dynamic and to mitigate students' difficulties in learning the content addressed. In this context, the present work approaches a teaching methodology that uses an automatic code broker and exploration of objective questions as a way to consolidate the subjects addressed during the theoretical classes, keeping the student informed about their development in the course. This methodology was applied and evaluated for some classes of the Engineering courses of the Federal University of Ouro Preto (UFOP) demonstrating the positive potential of students' learning and, consequently, their academic performance in the discipline.

Keywords: Programming. Logical reasoning. Automatic corrector.

Sumário

1 Introdução 1
1.1 Justificativa 2
1.2 Objetivos 2
1.3 Organização do Trabalho 3
2 Revisão Bibliográfica 4
2.1 Fundamentação Teórica 4
2.1.1 Corretores automáticos 4
2.1.2 Distância de Levenshtein 5
2.2 Trabalhos Relacionados 6
3 Desenvolvimento 8
3.1 Metodologia de ensino 8
3.2 O corretor automático de exercícios 9
3.2.1 Análise dinâmica 9
3.2.1.1 Pré-processamento 10
3.2.1.2 Correção 11
3.2.1.3 Geração do PDF 21
3.2.2 Análise Estática 22
4 Resultados 25
4.1 Análises da Avaliação Dinâmica 25
4.1.1 Análises da prova 1 25
4.1.2 Análises da prova 2 26
4.1.3 Análise do questionário de opinião 27
4.1.3.1 Perguntas aplicadas igualmente em ambos os semestres . . . 28
4.1.3.2 Perguntas relacionadas ao sistema de penalização vs bonificação 32
4.2 Análises da Avaliação Estática 33
4.3 Discussões 34
5 Considerações Finais 35
5.1 Trabalhos Futuros 35
5.2 Publicações Realizadas 35

Referências 36

Apêndices 37
APÊNDICE A Exemplo de um PDF gerado pelo corretor 38

1 Introdução

De acordo com as Diretrizes Curriculares Nacional dos Cursos de Graduação de Engenharia, disciplinas correlatas a Algoritmos e Programação devem estar presentes na grade curricular de diversas Engenharias. A inserção de tais disciplinas logo no ciclo básico ocorre, principalmente, devido à necessidade do desenvolvimento do raciocínio lógico do aluno e também, pela utilização dos conceitos dessas disciplinas ao longo do curso (Marcussi *et al.* , 2016).

Tais disciplinas são consideradas muito desafiadoras visto que exigem o desenvolvimento de estratégias de solução de problemas com base lógico-matemática. Na maior parte da vezes, o contato inicial com tais disciplinas é feito apenas na graduação, sujeitando o aluno a desenvolver o pensamento computacional em um curto espaço de tempo. Vários outros fatores podem dificultar ainda mais o processo de aprendizagem, como por exemplo: uso do inglês como principal idioma das linguagens de programação; desinteresse dos alunos por considerar estas disciplinas como desnecessárias para sua formação enquanto engenheiro; necessidade de se fazer essas disciplinas em conjunto com disciplinas consideradas "críticas", ou difíceis, como cálculo e álgebra; dentre outros.

Diversas ferramentas vêm sendo exploradas no intuito de aprimorar o processo ensino-aprendizagem em disciplinas relacionadas à programação e, ainda, automatizar o trabalho dos docentes com relação à correção de exercícios e emissão de *feedback* aos alunos. Em se tratando dos modelos de ensino atuais, com salas super lotadas, torna-se cada mais difícil atender e suprir as dificuldades individuais de cada aluno. Portanto, essas ferramentas surgem justamente como forma de atenuar a alta demanda que é repassada aos docentes. Além disso, elas podem também contribuir para identificar de maneira geral quais conteúdos devem ser melhor explorados devido às dificuldades dos alunos em resolver determinados exercícios.

Nesse trabalho, é explorada como a introdução de corretores automáticos de código bem como a utilização de questões objetivas para fixação do conteúdo teórico na metodologia de ensino de programação de computadores para alunos de engenharia contribuem para a aprendizagem do conteúdo abordado. Os corretores exploram tanto a corretude da saída gerada quanto o raciocínio lógico aplicado pelo aluno na construção de uma solução. A ideia é fazer com que os alunos consigam acompanhar sua evolução no curso mediante apresentação de *feedback* rápidos e explicativos e, também, atenuar os trabalhos dos docentes com relação à correção das inúmeras quantidades de tarefas e exercícios. Além disso, foram avaliados como a aplicação de sistemas de penalização/bonificação podem motivar os alunos na resolução de problemas.

Os resultados obtidos através de estudos de caso dão indícios de que a metodologia de ensino utilizada neste trabalho cumpre com seu objetivo ao auxiliar os alunos no desenvolvimento do raciocínio lógico e ao mantê-los motivados a realizarem as atividades propostas. Além disso, a

diversificação de ferramentas de correção dos exercícios contribuem no processo de aprendizagem na medida em que considera ao máximo o desenvolvimento do aluno na elaboração de uma solução.

O restante deste capítulo é organizado da seguinte forma. Na Seção 1.1 apresentamos justificativas para a realização deste trabalho. Na Seção 1.2 são definidos os objetivos gerais e específicos. Por fim, na Seção 1.3 apresentamos como este trabalho está organizado.

1.1 Justificativa

Disciplinas de programação transferem aos professores uma forte demanda de interação com os alunos pois faz-se necessário identificar e auxiliar as dificuldades de cada um. Muitas vezes, essas dificuldades não são detectadas e sanadas em tempo hábil, na maioria dos casos, por motivos didático-organizacionais. Como consequência, tem-se uma alta quantidade de reprovações e desistências, logo nos primeiros períodos do curso.

O alto nível de abstração exigido em disciplinas de programação causa desmotivação nos alunos, fator preponderante para as altas taxas de reprovações e desistências. Estudantes que iniciam uma graduação e não terminam geram desperdícios sociais, acadêmicos e econômicos. Em universidades não-privadas, tais desistências representam recursos públicos escassos que foram investidos e não obtiveram o devido retorno (Gonçalves *et al.* , 2013).

1.2 Objetivos

De maneira geral, o objetivo deste trabalho é avaliar como o uso de de ferramentas voltadas ao ensino de programação que emitem feedback rápidos e explicativos impactam no desenvolvimento do raciocínio lógico e também contribuem para o rendimento acadêmico dos alunos. São objetivos específicos:

- Definir uma metodologia de ensino que utilize ferramentas computacionais baseada em corretores automáticos e exploração de questões objetivas;
- Implementar um corretor automático de programas computacionais;
- Avaliar a aplicação de exercícios objetivos e práticos como forma de fixação do conteúdo teórico;
- Avaliar a efetividade e utilidade da metodologia para alunos de engenharia;

1.3 Organização do Trabalho

Este trabalho está organizado da seguinte forma: o Capítulo 2 apresenta uma breve revisão da literatura. No Capítulo 3 é descrito o desenvolvimento do trabalho, em que a metodologia proposta é detalhada. O Capítulo 4 apresenta resultados obtidos pela aplicação da metodologia em questão. Por fim, o Capítulo 5 apresenta as conclusões e trabalhos futuros.

2 Revisão Bibliográfica

As subseções a seguir apresentarão os conceitos necessários para compreensão do presente trabalho bem como alguns trabalhos relacionados à proposta aqui desenvolvida.

2.1 Fundamentação Teórica

As duas subseções a seguir descrevem conceitos necessários para compreensão deste trabalho, que são: corretores automáticos e distância de Levenshtein.

2.1.1 Corretores automáticos

Corretores automáticos de código, ou Juízes-online, são ferramentas acessíveis na Internet que permitem automatizar o processo de correção de exercícios de programação. Esses ambientes possuem a descrição de exercícios com entradas e saídas já pré-definidas e abordam os mais variados temas relacionados a programação.

O *URI Online Judge* é uma das mais tradicionais plataformas online que permite praticar a programação de maneira competitiva e gamificada. A plataforma possui uma vasta quantidade de exercícios que são agrupados de acordo com o assunto abordado nos mesmos. Dentre suas inúmeras funcionalidades, destacam-se:

- correção automática em tempo real das soluções submetidas;
- discussão dos problemas através de fóruns, possibilitando o compartilhamento de conhecimentos;
- aceitação de soluções em diversas linguagens de programação (como C, C++ e Python);
- uDebug integrado para verificar saídas para instâncias de entradas adicionais;
- separação dos problemas em categorias e níveis de dificuldades;
- aplicação de conceitos de gamificação, como rankings e sistema de recompensas;
- visualização dos erros encontrados durante a compilação e execução de uma solução;
- criação de competições simuladas;
- repositório das soluções submetidas.

Os critérios de avaliação impostos por esses corretores são muito exigentes. Os padrões de entrada e saída devem ser seguidos rigorosamente, caso contrário a solução é considerada incorreta, mesmo que o raciocínio desenvolvido pelo aluno seja válido.

2.1.2 Distância de Levenshtein

A Distância de Levenshtein, também chamada de Distância de Edição, é um algoritmo proposto pelo cientista russo Levenshtein (1966) que, dada duas strings, calcula o número mínimo de operações necessárias para se transformar uma string em outra. São consideradas como operações: a substituição, a inserção e a exclusão de caracteres. Esse algoritmo é muito utilizado quando se deseja verificar a similaridade entre duas strings. Observe que, quanto menor o número de operações necessárias, maior será o grau de similaridade entre as strings comparadas. O Algoritmo 1 é um pseudocódigo do cálculo da Distância de Levenshtein.

Algoritmo 1: Distancia de Levensthein

```
1  Função: distanciaLevenshtein(str1[0..tamStr1], str2[0..tamStr2])
2  início
3      Inteiro: tabela[0..tamStr1, 0..tamStr2]
4      Inteiro: i, j, custo
5      para i de 0 até tamStr1 faça
6          tabela[i, 0] ← i
7      fim
8      para j de 0 até tamStr2 faça
9          tabela[0, j] ← j
10     fim
11     para i de 1 até tamStr1 faça
12         para j de 1 até tamStr2 faça
13             se str1[i] = str2[j] então
14                 custo ← 0
15             fim
16             senão
17                 custo ← 2
18             fim
19             tabela[i, j] = menor( tabela[i-1, j] + 1, tabela[i , j-1] + 1, tabela[i-1, j-1] + custo)
20         fim
21     fim
22     retorna tabela[tamStr1,tamStr2]
23 fim
```

A função *distanciaLevenshtein* recebe como parâmetros duas strings de tamanhos não necessariamente iguais e retorna valor de custo mínimo para se transformar uma string na outra. Esse algoritmo utiliza a estratégia *bottom-up* da Programação Dinâmica, onde recálculos são evitados através do uso de uma tabela. Operações de inserção e exclusão possuem custo 1, enquanto operações de substituição possuem custo 2. Ao final da execução do algoritmo, a última cédula da tabela possui um valor, que indica o custo de se transformar uma string na outra

considerando o número mínimo de operações.

Neste trabalho, a Distância de Levensthein é explorada no desenvolvimento de uma ferramenta que realiza a correção automática de exercícios de programação, baseada na relação de similaridade entre a saída esperada e a saída obtida na execução de um programa feito pelo aluno.

2.2 Trabalhos Relacionados

Diversos trabalhos introduzem em suas metodologias de ensino ferramentas computacionais no intuito de auxiliar o processo ensino-aprendizagem em disciplinas introdutórias de programação. A seguir, serão apresentadas algumas dessas ferramentas bem como os resultados observados após implantação das mesmas.

A proposta desenvolvida por Jesus *et al.* (2018) parte do pressuposto de que as mensagens de erro emitidas pelos corretores automáticos não são intuitivas. O problema se agrava quando o estudante não possui conhecimento em inglês, visto que grande parte das linguagens de programação emitem *feedback* nesse idioma. Para contornar esse problema, os autores desenvolveram uma ferramenta que, aliada a um corretor automático, é capaz de emitir mensagens de erros mais sugestivas. Os resultados revelam indícios de que a apresentação de mensagens amigáveis podem melhor guiar os alunos para a correção dos erros. Entretanto, a ferramenta é capaz de detectar e auxiliar apenas em erros relacionados à sintaxe da linguagem abordada.

Na pesquisa conduzida por Raabe *et al.* (2015) é apresentado o framework de um corretor automático que combina análise estática e dinâmica dos códigos para emitir *feedback* aos alunos relacionados aos erros encontrados. Apesar da ferramenta ter sido avaliada utilizando uma amostra relativamente pequena (23 alunos), constatou-se que o corretor gerou um impacto positivo na dinâmica de resolução e depuração das soluções pelos alunos, entretanto revelou a necessidade de se utilizar mais casos de testes estáticos durante a avaliação.

O URI Online Judge é uma ferramenta online que permite praticar a programação e compartilhar o conhecimento, além de fornecer *feedback* em tempo real para os usuários que submeterem exercícios em sua plataforma. Berssanette & Carlos (2018) apresentam uma metodologia de ensino para a disciplina de Algoritmos e Lógica em que utilizam o portal do URI Online Judge aliado a técnica de PBL (método de ensino centrado no aluno, consistindo, basicamente, na resolução de exercícios como forma de aprendizagem). Setenta e oito por cento dos alunos que cursaram a disciplina seguindo essa metodologia obtiveram índices satisfatórios.

Piekarski *et al.* (2015) apresentaram uma metodologia de ensino baseada no emprego de elementos presentes nas maratonas de programação (sistema de ranqueamento, resolução de problemas e competições simuladas) como forma de desenvolvimento do raciocínio lógico e do trabalho colaborativo. A correção dos exercícios é feita de maneira automática e em tempo

real, entretanto os alunos devem seguir rigorosamente os padrões de entradas pré-definidos na descrição do problema. Além de desenvolver habilidades nas linguagens de programação, a introdução dessa metodologia estimulou os alunos a resolverem problemas aplicados nas competições, sendo esses os principais resultados alcançados.

Uma das apostas do século XXI na área educacional envolve o uso da gamificação para engajar os alunos a atingirem um determinado objetivo. Sgoti & Mill (2018) exemplificam o uso da gamificação através da criação de uma lista de exercícios para a disciplina de algoritmos utilizando sistemas de recompensa e bonificação. Além de motivar os alunos, a lista gamificada alcançou um maior índice de aproveitamento na resolução dos exercícios quando comparada à lista de exercícios convencional. Outros trabalhos, como os desenvolvidos por Carreño-León *et al.* (2018) e Silva *et al.* (2016), também fizeram uso da gamificação como parte da metodologia de ensino e obtiveram resultados similares. A grande dificuldade nestes trabalhos se referem ao processo de correção. Os exercícios eram corrigidos manualmente pelo professor, gerando sobrecarga de trabalho para o mesmo.

Boa parte dos trabalhos acima citados fazem uso de ferramentas online que fornecem *feedback* em tempo real aos alunos, permitindo que eles identifiquem seus erros e os corrijam o mais rápido possível. Entretanto, esses corretores são muito rigorosos com relação à padrões de entrada e saída dos problemas. O aluno pode, por exemplo, perder toda uma questão por simplesmente colocar um espaço em branco a mais na saída da sua solução. A ideia da metodologia aqui proposta é valorizar ao máximo o raciocínio lógico do aluno, tornando o aprendizado mais flexível e didático.

3 Desenvolvimento

As subseções a seguir descreverão a metodologia de ensino utilizada nas disciplinas de programação para alunos dos cursos de engenharia da Universidade Federal de Ouro Preto (UFOP) bem como uma ferramenta desenvolvida como forma de apoio à essa metodologia.

3.1 Metodologia de ensino

A disciplina de Programação de Computadores I é ofertada para alunos de variados cursos de engenharia, sendo dividida em aulas teóricas e aulas práticas. As aulas teóricas seguem a seguinte dinâmica: primeiramente, o professor apresenta um problema do cotidiano que pode ser computacionalmente resolvido e conduz os alunos a refletirem de qual maneira eles o resolveriam. Após isso, são apresentados conceitos necessários para a resolução do problema bem como uma possível solução para o mesmo em uma linguagem de programação. A metodologia aqui proposta utiliza uma nova abordagem para as aulas práticas, que segue a seguinte dinâmica: são propostos exercícios de fixação e exercícios práticos, ambos relacionados ao conteúdo apresentado nas aulas teóricas. Diferentemente das aulas teóricas, em que as aulas são ministradas em salas de aula convencionais, as aulas práticas acontecem nos laboratórios de informática onde os alunos realizam os exercícios propostos.

Conforme dito, as aulas práticas são segmentadas em exercícios de fixação e exercícios práticos. O principal objetivo dessas aulas é fazer com que os alunos absorvam, da melhor maneira possível, conceitos apresentados durante as aulas teóricas. A primeira parte das aulas práticas corresponde à aplicação de exercícios de fixação que são questões objetivas vinculadas ao Moodle (Ambiente Virtual de Aprendizagem) e que permitem avaliar o entendimento do aluno com relação ao pensamento lógico na resolução de problemas, à estrutura sintática da linguagem de programação e à execução de programas. A avaliação dessas questões é feita de maneira totalmente automática através de recursos oferecidos pelo Moodle. A segunda parte das aulas práticas correspondente à codificação de soluções para os diversos problemas apresentados. Para se avaliar este tipo de questão, foi desenvolvida uma ferramenta que permite fazer a correção automática baseando-se em critérios pré definidos. A próxima subseção descreverá essa ferramenta.

Nos exercícios práticos espera-se que o aluno seja capaz de:

1° Ler atentamente o problema;

2° Definir quais serão as variáveis de entrada e de saída da sua solução;

3° Definir quais os passos para resolução do problema bem como a ordem em que estes deverão ser feitos;

4° Apresentar as saídas conforme lhe é solicitado na descrição do problema;

No primeiro semestre em que a metodologia foi aplicada, foram definidas penalizações graduais para aqueles alunos que não entregavam as atividades durante a aula. Entretanto, após a análise de um questionário aplicado a estes alunos constatou-se que as penalizações não os motivavam a realizar as atividades. Assim, no intuito de contornar esse problema, no segundo semestre foram definidas bonificações nas notas para aqueles que entregavam as atividades durante a aula. Adicionalmente, no segundo semestre os prazos para entrega das atividades foram maiores, cerca de 4 dias após a aula prática. No primeiro semestre o prazo era encerrado no mesmo dia da aula prática.

Além das atividades práticas, os alunos realizam duas provas teóricas que abordam todo o conteúdo apresentado na sala de aula. As questões propostas nessas provas se assemelham muito ao estilo dos exercícios abordados nas aulas práticas. A grande diferença é que as atividades práticas são realizadas no computador enquanto as provas são feitas no papel.

3.2 O corretor automático de exercícios

Os corretores automáticos disponíveis na literatura avaliam apenas a estrutura algorítmica de uma solução e/ou avaliam aspectos gerais da execução do programa. Especificamente em relação à saída originada pela execução da solução do aluno, eles apenas avaliam se ela está exatamente igual à saída esperada, eventualmente apresentando um percentual geral de similaridade. Do ponto de vista pedagógico, este tipo de *feedback* ao aluno limita consideravelmente a capacidade de auxilia-lo no aprendizado e em valorizar os aspectos mais relevantes da solução do problema em relação aos conceitos de programação em avaliação.

Existem dois tipos de análises quando se deseja fazer a correção de um programa: a dinâmica e a estática (Raabe *et al.* , 2015). Na análise dinâmica, o código do aluno é executado e funciona como uma "caixa preta" em que dada uma entrada é produzida uma saída decorrente da execução do programa. Já na análise estática, apenas o código do aluno é analisado. Logo, ele é avaliado em termos de complexidade e de sequenciamento correto dos comandos. O corretor de exercícios aqui proposto realiza tanto a análise dinâmica quanto a estática. Ambas análises serão detalhadas a seguir.

3.2.1 Análise dinâmica

Conforme dito anteriormente, na análise dinâmica a solução do aluno é avaliada de acordo com as saídas e critérios pré-definidos. Para tanto, o corretor automático segue as etapas de execução disponíveis na Figura 3.1 e detalhadas na subseções a seguir.

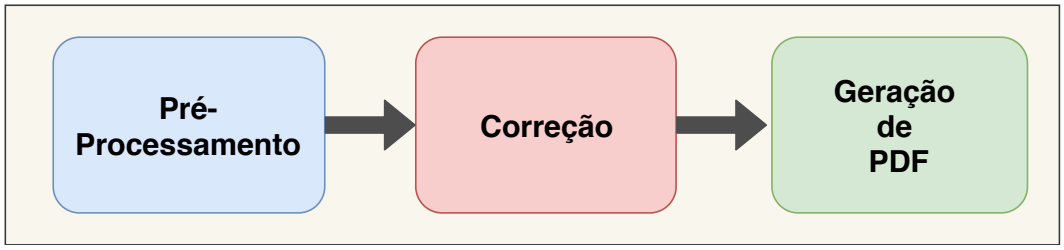

Figura 3.1 – Fluxo de execução do corretor automático

3.2.1.1 Pré-processamento

Antes de iniciar a correção propriamente dita é necessário definir casos de teste que serão utilizados na validação de cada solução submetida pelo aluno. Cada caso de teste é composto por dois arquivos, que são:

- **Arquivos de entrada**: arquivos texto que simulam a entrada de dados para um programa. Geralmente, para se testar uma determinada implementação, os alunos realizam essa entrada de dados manualmente (via teclado). Entretanto, para que a correção seja realizada de maneira automática é necessário que a entrada de dados seja independente do usuário.
- **Arquivos de critérios**: arquivos texto que definem a saída esperada para cada arquivo de entrada bem como as pontuações associadas a cada linha de saída. Os critérios são definidos levando em consideração que existem elementos presentes na saída que são mais relevantes que outros. Por exemplo, em questões que possuem em sua saída mensagens de texto padrão e cálculos matemáticos, a presença deste último deve ser melhor pontuada, uma vez que mensagens de texto padrão não exigem nenhum raciocínio lógico para implementação, apenas devem ser exibidas. A estrutura de um arquivo de critério é apresentado na Figura 3.2. Note que as linhas são particionadas. Esse processo é feito justamente para avaliar separadamente saídas de texto padrão de cálculos matemáticos, por exemplo.

Tanto os arquivos de entrada quanto os arquivos de critérios são armazenados dentro de um mesmo diretório e possuem nome padrão. Os arquivos de entrada são nomeados como “questaoX.entrada.Y” e os arquivos de critérios nomeados como “questaoX.criterios.Y”. Em ambos os casos, X representa o número da questão associada e Y o número do caso de teste. A Figura 3.3 apresenta um exemplo dos arquivos necessários e gerados na etapa de pré-processamento. Na Figura 3.3 (a) tem-se a descrição do problema que deve ser resolvido pelo aluno. Na Figura 3.3 (b) é dado um exemplo que ilustra a execução completa de uma solução correta, desde a entrada de dados até a saída do programa. A Figura 3.3 (c) mostra as entradas que serão utilizadas para validar a solução do aluno. Na Figura 3.3 (d) tem-se os critérios pré-estabelecidos para avaliação. Note que os cálculos matemáticos possuem pontuação maior quando comparados às mensagens de texto padrão.

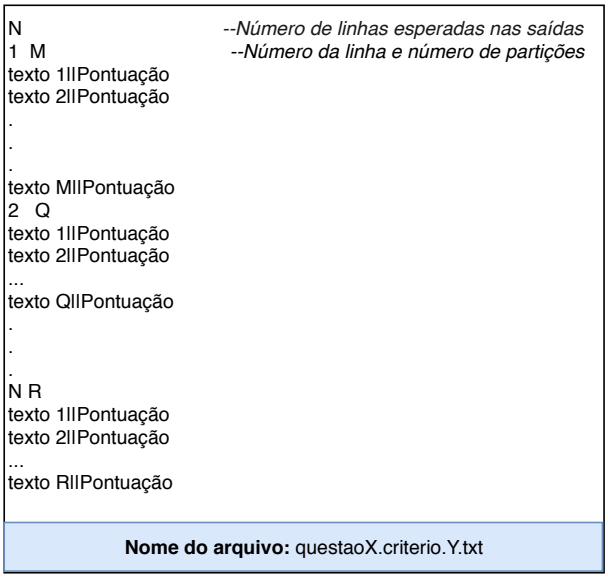

Figura 3.2 – Estrutura do arquivo de critério

3.2.1.2 Correção

As soluções implementadas para cada problema proposto nas aulas práticas são submetidas pelos aluno via Moodle. Ao serem coletados, estes arquivos já estão separados em diretórios, em que cada diretório é nomeado com o nome do aluno e possui as implementações submetidas por este.

Inicialmente, o corretor percorre cada um dos diretórios e procura os arquivos de solução do aluno. Estes arquivos também possuem um nome padrão que apresenta a seguinte forma: "questaoX.sce", em que X representa o número da questão submetida para avaliação. A solução enviada pelo aluno é executada utilizando todas as entradas pré-definidas para essa questão e as saídas obtidas são armazenadas em arquivos texto temporários. Esse processo é realizado automaticamente através de uma chamada ao sistema utilizando redirecionamento de entrada e saída.

Para se medir o grau de similaridade (S) entre a saída esperada pelo programa e a saída obtida pelo aluno, utiliza-se uma função de comparação que, baseada na Distância Levenshtein retorna um valor P, representando a porcentagem de similaridade entre as strings comparadas. A Figura 3.4 apresenta como é realizado o processo para se obter a pontuação do aluno.

Na Figura 3.4 (a) tem-se a descrição do problema bem como a saída esperada para uma entrada pré-definida. A Figura 3.4 (b) mostra os arquivos definidos na etapa de pré-processamento. Note que a pontuações foram distribuídas considerando maior relevância para o cálculo matemático, que neste caso, consistia em elevar um número ao quadrado. Na Figura 3.4 (c) é apresentada as comparações de cada partição do arquivo de critério com a saída obtida bem como o percentual de similaridade entre elas. Multiplicando-se esse percentual pelo valor definido para a correspondente partição, obtêm-se a pontuação para a referida partição. A soma de todas essas

Questão 1
Considere as expressões a seguir:

1. $\frac{x^2y^3}{(x-y)^2}$
2. $\frac{1}{x^2-y} - e^{-4x} + \sqrt[3]{\frac{35}{y}}\sqrt{xy}$
3. $\frac{4}{3}\pi\, sen(x^2) - 1$

Implemente um programa em Scilab que faça a leitura dos valores de x e y e depois calcule e imprima os resultados das expressões.

a) Descrição do problema

Exemplo:
Digite o valor de x: 3
Digite o valor de y: 4
Valor da expressão 1: 576
Valor da expressão 2: 7.33827
Valor da expressão 3: 0.726278

b) Exemplo de execução

```
3
4
```

Nome do arquivo: questao1.entrada.1.txt

c) Arquivo de entrada

```
3
1 2
Valor da expressão 1:||0.333
576||3.0
2 3
Valor da expressão 2:||0.333
7.||2.0
.33827||1.0
3 3
Valor da expressão 3:||0.333
0.||1.0
.726278||2.0
```

Nome do arquivo: questao1.criterios.1.txt

d) Arquivo de critério

Figura 3.3 – Arquivos utilizados na etapa de pré-processamento

pontuações corresponde à pontuação final do aluno para o caso de teste em questão.

Existem situações em que a solução do aluno pode ser altamente penalizada mesmo sua lógica estando correta. Essas situações correspondem, na maioria dos casos, à não formatação correta das mensagens de saída. Com o objetivo de valorizar ao máximo o raciocínio lógico do aluno, cada arquivo de saída gerado na execução da solução proposta é submetida à determinados procedimentos, os quais são descritos a seguir.

Exercício
Escreva um programa que leia um número inteiro qualquer e imprima este mesmo número elevado ao quadrado.

Entrada de Dados	Saída Esperada
DIGITE UM NÚMERO INTEIRO: 8	*Linha 1:* 8 AO QUADRADO É IGUAL A 64

(a) Descrição do problema.

Arquivo de entrada	Arquivo de critério
8	1 1 3 8||1 AO QUADRADO É IGUAL A||0.5 64||8.5

(b) Arquivos definidos no pré-processamento.

Saída obtida
8 ELEVADO AO QUAD É IGUAL A 64

String comparada com a saída obtida	Similaridade	Pontuação
8	100%	1,0 X 1,0
AO QUADRADO É IGUAL A	81%	0,81 X 0,5
64	100%	1,0 X 8,5
	TOTAL	**9,9**

(c) Cálculo da pontuação baseado na similaridade das strings.

Figura 3.4 – Cálculo da pontuação obtida pelo aluno.

1° **Elimina-se as linhas em branco**.
As linhas em branco podem causar desordenação nas mensagens fazendo com que a saída do aluno seja desconsiderada em situações em que seu raciocínio lógico está correto. A Figura 3.5 apresenta um exemplo que ilustra a penalização causada pela inclusão de linhas em branco na saída bem como os efeitos produzidos pela eliminação dessas.

Na Figura 3.5 (a) temos a descrição do problema juntamente com a saída esperada para uma entrada pré-definida. A Figura 3.5 (b) mostra a pontuação que o corretor atribuía à saída do aluno quando não se eliminava as linhas em branco. Note que a pontuação nesse caso de teste foi obtida ao comparar a primeira linha esperada com a primeira linha obtida. A segunda linha esperada foi comparada com a segunda linha obtida, que neste caso era uma linha em branco, fazendo com que o aluno não recebesse nenhuma pontuação referente à linha dois. Como a saída esperada era composta por duas linhas e o corretor se baseia na correlação entre linhas, todas as linhas impressas além da quantidade esperada foram descartadas. A Figura 3.5 (c) mostra a reordenação da saída após eliminação das linhas em branco. Observe como essa simples modificação na saída possibilitou uma melhor valorização da solução do aluno.

Exercício
Pode-se calcular a área e o perímetro de um triângulo, dados os comprimentos dos seus lados - s1, s2 e s3 - de acordo com as seguintes fórmulas:

$$area = \sqrt{s \times (s - s_1) \times (s - s_2) \times (s - s_3)}$$

onde:

$$s = \frac{s_1 + s_2 + s_3}{2}$$

$$perímetro = s_1 + s_2 + s_3$$

Escreva um programa que leia os comprimentos dos lados de um triângulo - s 1, s 2 e s 3 - e imprima o perímetro e a área do triângulo, conforme o exemplo de execução abaixo.

Entrada de Dados	Saída Esperada
DIGITE O LADO 1 DO TRIÂNGULO (m): 10 DIGITE O LADO 2 DO TRIÂNGULO (m): 10 DIGITE O LADO 3 DO TRIÂNGULO (m): 8	*Linha 1:* PERÍMETRO DO TRIÂNGULO = 28 m *Linha 2:* ÁREA DO TRIÂNGULO = 36.6606 m^2

(a) Descrição do problema

Saída Esperada	Saída Obtida	Pontuação
Linha 1: PERÍMETRO DO TRIÂNGULO = 28 m *Linha 2:* ÁREA DO TRIÂNGULO = 36.6606 m^2	*Linha 1:* PERÍMETRO DO TRIÂNGULO = 28 m *Linha 2:* *Linha 3:* ÁREA DO TRIÂNGULO = 36.6606 m^2	5,0

(b) Pontuação obtida sem descartar linhas em branco.

Saída obtida	Saída Obtida	Pontuação
Linha 1: PERÍMETRO DO TRIÂNGULO = 28 m *Linha 2:* ÁREA DO TRIÂNGULO = 36.6606 m^2	*Linha 1:* PERÍMETRO DO TRIÂNGULO = 28 m *Linha 2:* ÁREA DO TRIÂNGULO = 36.6606 m^2	10,0

(c) Pontuação obtida descartando-se as linhas em branco.

Figura 3.5 – Efeitos causados pela eliminação das linhas em branco.

2° **Elimina-se a acentuação de caracteres**.
Uma caractere sem acentuação correta pode provocar penalidades significativas à solução do aluno embora ele tenha tido um raciocínio correto para resolver o problema. A Figura 3.6 apresenta um exemplo que ilustra a penalização causada pela não acentuação correta dos caracteres da saída e, também, como uma solução pode ser melhor valorizada ao se ignorar essas acentuações.

Na Figura 3.6 (a) temos a descrição do problema juntamente com a saída esperada para uma entrada pré-definida. A Figura 3.6 (b) mostra como a solução do aluno foi consideravelmente penalizada por não acentuar corretamente as palavras. Neste caso, ao comparar a primeira linha esperada com a primeira linha obtida, a função de comparação retornou valor P = 65, expressando que as linhas comparadas são 65% similares. A Figura 3.6 (c) apresenta a nova pontuação do aluno ao ignorar a acentuação dos caracteres.

Exercício

"Papai Noel está brincando com seus duendes para entretê-los durante a véspera do Natal. A brincadeira consiste nos elfos escreverem números em pedaços de papel e colocarem no gorro do Papai Noel. Após todos terminarem de colocar os números Noel sorteia um papel e aquele número representa quantos "Hô" o Noel deve falar.

Seu trabalho é ajudar o Papai Noel montando uma solução que mostre todos os "Hô" que ele deve falar dado o número sorteado." - Por Lucas Campesatto, URI Online Judge

Entrada de Dados	Saída Esperada
DIGITE UM NÚMERO NATURAL: 8	*Linha 1:* Hô Hô Hô Hô Hô Hô Hô Hô

(a) Descrição do problema

Saída Esperada	Saída Obtida	Pontuação
Linha 1: Hô Hô Hô Hô Hô Hô Hô Hô	*Linha 1:* Ho Ho Ho Ho Ho Ho Ho Ho	6,50

(b) Pontuação obtida ao não desconsiderar as acentuações das palavras.

Saída Esperada	Saída Obtida	Pontuação
Linha 1: Ho Ho Ho Ho Ho Ho Ho Ho	*Linha 1:* Ho Ho Ho Ho Ho Ho Ho Ho	10,0

(c) Pontuação obtida desconsiderando-se as acentuações das palavras.

Figura 3.6 – Efeitos causados pela eliminação das acentuações dos caracteres

3° **Elimina-se caracteres especiais**.

Os caracteres especiais, sobretudo em exercícios de programação, são requisitados quando se deseja construir uma saída mais atrativa, não possuindo, na maioria dos casos, relação com a lógica à ser desenvolvida. Portanto, eles também podem ser descartados. A Figura 3.7 apresenta um exemplo completo que mostra os efeitos em uma correção ao se manter e ao se descartar caracteres especiais.

Na Figura 3.7 (a) temos a descrição do problema juntamente com a saída esperada para uma entrada pré-definida. A Figura 3.7 (b) mostra que a função de comparação considerou a saída obtida apenas 47% similar à saída esperada. Isso aconteceu porque ao invés de imprimir o caractere '*' foi impresso o caractere '-' e, como boa parte da saída era formada pela presença desse primeiro caractere, a solução do aluno foi consideravelmente penalizada. Ao desconsiderar caracteres especiais, na Figura 3.7 (c), obteve-se uma melhor valorização da solução do aluno, uma vez que esses caracteres não eram mais relevantes na correção.

Exercício
Escreva um programa em Scilab que leia um número natural e imprima a mensagem "*****É PAR*****" caso o número digitado seja par ou "*****É ÍMPAR*****" caso o número digitado seja ímpar. O programa deve ser encerrado automaticamente quando um número não natural for digitado.

Entrada de Dados	Saída Esperada
DIGITE UM NÚMERO NATURAL: 8 DIGITE UM NÚMERO NATURAL: 11 DIGITE UM NÚMERO NATURAL: -5	*Linha 1:* *****É PAR!***** *Linha 2:* *****É ÍMPAR!***** *Linha 3:* PROGRAMA FINALIZADO!

(a) Descrição do problema

Saída Esperada	Saída Obtida	Pontuação
Linha 1: *****É PAR!***** *Linha 2:* *****É ÍMPAR!***** *Linha 3:* PROGRAMA FINALIZADO!	*Linha 1:* -----É PAR!----- *Linha 2:* -----É ÍMPAR!----- *Linha 3:* PROGRAMA FINALIZADO!	4,70

(b) Pontuação obtida ao não desconsiderar os caracteres especiais.

Saída Esperada	Saída Obtida	Pontuação
Linha 1: É PAR! *Linha 2:* É ÍMPAR! *Linha 3:* PROGRAMA FINALIZADO!	*Linha 1:* É PAR! *Linha 2:* É ÍMPAR! *Linha 3:* PROGRAMA FINALIZADO!	10,0

(c) Pontuação obtida ao descartar os caracteres especiais.

Figura 3.7 – Efeitos causados pela eliminação dos caracteres especiais

4° **Elimina-se espaços em branco**.

É muito comum que os alunos imprimam espaços em branco a mais ou a menos durante a elaboração da saída de uma solução. Entretanto, essa desconformidade entre a quantidade de espaços impressos e a quantidade de espaços esperada pode gerar penalizações significativas durante a correção. Por esse motivo e devido à irrelevância das presença/ausência dos espaços, optou-se por eliminá-los. A Figura 3.8 apresenta um exemplo que mostra as consequências de se manter e de se excluir espaços em branco.

Na Figura 3.8 (a) tem-se a descrição do problema juntamente com a saída esperada para uma entrada pré-definida. A Figura 3.7 (b) mostra como a inclusão de espaços em branco além do solicitado afeta diretamente na pontuação. Neste caso, de acordo com a função de comparação, a saída obtida era 55% similar a saída esperada. A Figura 3.7 (c) apresenta as linhas de saída sem considerar os espaços em branco entre as palavras bem como a pontuação obtida por se aplicar essa filtragem.

Exercício

"Escreva um programa que repita a leitura de uma senha até que ela seja válida. Para cada leitura de senha incorreta informada, escrever a mensagem "Senha Invalida". Quando a senha for informada corretamente deve ser impressa a mensagem "Acesso Permitido" e o algoritmo encerrado. Considere que a senha correta é o valor 2002." - Por Neilor Tonin, URI Online Judge

Entrada de Dados	Saída Esperada
2200 1020 2002	*Linha 1:* Senha Invalida *Linha 2:* Senha Invalida *Linha 3:* Acesso Permitido

(a) Descrição do problema

Saída Esperada	Saída Obtida	Pontuação
Linha 1: Senha Invalida *Linha 2:* Senha Invalida *Linha 3:* Acesso Permitido	*Linha 1:* Senha Invalida *Linha 2:* Senha Invalida *Linha 3:* Acesso Permitido	5,50

(b) Pontuação obtida ao não desconsiderar os espaços em branco.

Saída Esperada	Saída Obtida	Pontuação
Linha 1: SenhaInvalida *Linha 2:* SenhaInvalida *Linha 3:* AcessoPermitido	*Linha 1:* SenhaInvalida *Linha 2:* SenhaInvalida *Linha 3:* AcessoPermitido	10,0

(c) Pontuação obtida desconsiderando-se os espaços em branco.

Figura 3.8 – Efeitos causados pela eliminação dos espaços em branco.

5° **Elimina-se os sinais de pontuação, exceto o ponto final**.

A presença e a falta dos sinais de pontuação não é considerada relevante na avaliação da lógica desenvolvida pelo aluno, por isso são descartados. O motivo de se manter o ponto final é porque em situações que envolvem cálculos matemáticos com números reais, por exemplo, é necessário avaliar separadamente a parte inteira da parte fracionária. A Figura 3.9 apresenta um exemplo que mostra as consequências de se manter e de se excluir os sinais de pontuação.

Na Figura 3.9 (a) tem-se a descrição do problema juntamente com a saída esperada para uma entrada pré-definida. A Figura 3.9 (b) mostra que, como o sinal de exclamação fazia parte da mensagem de saída, a função de comparação considerou as linhas da saída obtida apenas 62, 0% similares às linhas da saída esperada. A Figura 3.9 (c) apresenta a nova pontuação obtida quando não consideramos os sinais de pontuação. Observe que nesse exercício, o importante era saber distinguir um número primo de um não primo. Logo, ao desconsiderar os sinais de pontuação, a solução do aluno pode ser melhor valorizada.

Exercício

Priscila é uma menina que adora números primos. Acontece que existem números tão grandes, mas tão grandes, que Priscila não consegue descobrir se eles são primos ou não. Ajude Priscila nessa tarefa! escreva um programa que leia um número natural e imprima "**É PRIMO!!!!!!!!!!**" caso o número digitado seja primo e **"NÃO É PRIMO !!!!!!!!!!**" caso contrário. A entrada de dados termina quando um número negativo for digitado.

Entrada de Dados	Saída Esperada
DIGITE UM NÚMERO: 307 DIGITE UM NÚMERO: 450 DIGITE UM NÚMERO: -2	*Linha 1:* É PRIMO!!!!!!!!!! *Linha 2:* NÃO É PRIMO !!!!!!!!!!

(a) Descrição do problema

Saída Esperada	Saída Obtida	Pontuação
Linha 1: É PRIMO!!!!!!!!!! *Linha 2:* NÃO É PRIMO !!!!!!!!!!	*Linha 1:* É PRIMO *Linha 2:* NÃO É PRIMO	6,20

(b) Pontuação obtida ao não desconsiderar sinais de pontuação.

Saída Esperada	Saída Obtida	Pontuação
Linha 1: É PRIMO *Linha 2:* NÃO É PRIMO	*Linha 1:* É PRIMO *Linha 2:* NÃO É PRIMO	10,0

(c) Pontuação obtida desconsiderando-se os sinais de pontuação.

Figura 3.9 – Efeitos causados pela eliminação dos sinais de pontuação.

6° **Todo texto é convertido para letras minúsculas**.

Salvo casos especiais, pode-se considerar que uma mensagem escrita em letra maiúscula não é diferente dessa mesma mensagem escrita em letra minúscula, por isso todos os caracteres alfabéticos são substituídos por seu corresponde em minúsculo. A Figura 3.10 mostra uma situação em que praticamente toda solução do aluno foi desconsidera pelo fato da mensagem de saída não ter sido impressa em letras maiúsculas. Apresenta também, como é possível contornar essa situação e valorizar o raciocínio desenvolvido pelo aluno.

Na Figura 3.10 (a) tem-se a descrição do problema juntamente com a saída esperada para uma entrada pré-definida. A Figura 3.10 (b) mostra que a função de comparação considerou as linhas comparadas apenas 5% similares. Isso porque, neste caso, apenas o espaço em branco que separa as duas palavras eram iguais em ambas saídas. A Figura 3.10 (c) mostra toda a saída sendo convertida para letras minúsculas e como essa mudança fez com que o raciocínio do aluno fosse amplamente valorizado.

Exercício
Escreva um programa que leia os comprimentos dos dois lados de um retângulo, considerando que o lado maior será digitado primeiro, e compare a área desse retângulo com a área do círculo cujo diâmetro tem medida igual à do maior lado do retângulo. O programa deve imprimir uma mensagem indicando se a área do círculo é ou não é maior do que a do retângulo, conforme ilustrado nos exemplo a seguir.

Entrada de Dados	Saída Esperada
DIGITE O PRIMEIRO ÂNGULO INTERNO: 45 DIGITE O SEGUNDO ÂNGULO INTERNO: 80 DIGITE O TERCEIRO ÂNGULO INTERNO: 55	*Linha 1:* TRIÂNGULO ACUTÂNGULO

(a) Descrição do problema

Saída Esperada	Saída Obtida	Pontuação
*Linha 1:*TRIÂNGULO ACUTÂNGULO	*Linha 1:* triângulo acutângulo	0,5

(b) Pontuação obtida ao não transformar letras maiúsculas em minúsculas.

Saída Esperada	Saída Obtida	Pontuação
Linha 1: triângulo acutângulo	*Linha 1:* triângulo acutângulo	10,0

(c) Pontuação obtida ao transformar letras maiúsculas em minúsculas.

Figura 3.10 – Efeitos causados pela transformação de todo texto para letras minúsculas.

Todo esse processo é feito no intuito de valorizar a lógica por trás da solução apresentada. Isso porque os alunos, muito deles recém concluintes do ensino médio, não estão habituados a essa nova forma de pensamento em que padrões especificados devem ser seguidos à risca. Conforme pode ser notado nos exemplos, mesmo quando a lógica existente para resolução do problema estava correta, o corretor desconsiderava em muito a solução do aluno pelo fato do formato das mensagens de saída não ter sido seguido rigorosamente. Essas penalizações podem acabar os desmotivando.

Realizados todos os processos acima descritos é possível iniciar a verificação de proximidade entre a saída esperada e a saída obtida pelo aluno. Conforme dito anteriormente, para se medir o grau de similaridade (S) entre as duas saídas, utiliza-se uma função baseada no algoritmo da Distância Levenshtein. Essa função retorna um valor P, que representa a porcentagem de similaridade entre as strings comparadas. É definido, também, um Limiar (L) como parâmetro para aceitação ou rejeição de proximidade entre duas strings. No corretor, o grau de similaridade entre duas strings pode ser definido como:

$$\mathrm{S} = \begin{cases} 0, & P < L \\ P, & \text{caso contrário} \end{cases}$$

Definir esse limiar é importante para não se atribuir pontuações indevidas à solução do aluno. Observe a Figura 3.11 que mostra como o aluno seria consideravelmente bonificado mesmo resolvendo de maneira totalmente incorreta o exercício e, também, como a definição de

um limiar ajudaria a resolver esse problema. Na Figura 3.11 (a) mostra a descrição do problema bem como a saída esperada para uma entrada pré-definida. Na Figura 3.11 (b) mostra que a linha da saída obtida era 67% similar à linha da saída esperada, logo o aluno foi bonificado com 6,70 pontos por essa similaridade, mesmo a resposta estando totalmente incorreta. Na Figura 3.11 (c) foi definido um limiar L = 90%, fazendo com que a resposta do aluno fosse rejeitada.

Exercício

Escreva um programa que leia um número referente ao mês do ano e imprima a mensagem "UHULL, O NATAL ESTÁ CHEGANDO!" caso o número digitado corresponda ao mês de Dezembro ou "VISH, O NATAL ESTÁ LONGE!", caso contrário.

Entrada de Dados	Saída Esperada
12	*Linha 1:* UHULL, O NATAL ESTÁ CHEGANDO!

(a) Descrição do problema

Saída Esperada	Saída Obtida	Pontuação
Linha 1: UHULL, O NATAL ESTÁ CHEGANDO!	*Linha 1:* VISH, O NATAL ESTÁ LONGE!	6,70

(b) Pontuação obtida sem considerar um limiar.

Saída Esperada	Saída Obtida	Pontuação
Linha 1: UHULL, O NATAL ESTÁ CHEGANDO!	*Linha 1:* VISH, O NATAL ESTÁ LONGE!	0,00

(c) Pontuação obtida considerando-se um limiar.

Figura 3.11 – Efeitos causados pela definição de um limiar.

Existem situações em que a proposta de solução do aluno possui erros lógicos ou sintáticos. Nestes casos, o corretor emite mensagens específicas, baseadas nos problemas encontrados. A tabela a seguir apresenta as possíveis mensagens emitidas e os problemas relacionados a elas.

Mensagem	**Descrição**
Erro de sintaxe.	A solução possui erros ligados à estrutura sintática da linguagem.
Erro de execução.	Possivelmente o programa encerrou sem antes processar todas as linhas do arquivo de entrada. Ou mesmo, fez acesso a posições inválidas da memória.
Seu programa entrou em loop infinito.	Antes de se iniciar uma correção é definido um tempo máximo para que o programa gere uma saída. Logo, se o tempo para execução terminou e o programa não se encerrou é detectado um loop infinito.
Você não utilizou função para resolver a questão.	Trata-se de exercícios específicos em que é obrigatório o uso de funções para resolvê-los. Somente aqui é realizada uma análise estática do código.
O arquivo de solução não foi encontrado.	O aluno não submeteu sua solução para correção.

Tabela 3.1 – Mensagens padronizadas emitidas pelo corretor.

A detecção de funções é feita percorrendo o arquivo que contém a solução do aluno e identificando palavras reservadas da linguagem que devem ser utilizadas quando se deseja definir uma função. Apesar de ser uma abordagem ingênua, o corretor consegue afirmar com exatidão se o aluno definiu ou não uma função para construir sua solução.

As saídas emitidas pela execução das soluções de cada aluno bem como a pontuação atribuída a estas saídas são armazenadas em um arquivo texto temporário. Esse arquivo é utilizado para gerar o PDF com a correção para o aluno.

3.2.1.3 Geração do PDF

Após a correção dos exercícios, inicia-se a etapa de geração do PDF. Essa etapa corresponde a formalização de todas as informações referentes à correção da atividade prática. O arquivo PDF gerado possui as seguintes informações:

- Data da atividade proposta;
- Número da aula;
- Assunto abordado;
- Número dos exercícios propostos;
- Tabela que sintetiza o resultado geral do aluno na atividade;
- Bonificações por exercícios entregues no horário da aula;

- Descrição dos casos de teste utilizados na correção de cada questão bem como a saída esperada, a saída obtida e a pontuação do aluno;

No Apêndice A é disponibilizado um exemplo contendo o arquivo gerado pela correção da atividade prática de um aluno. Cada arquivo gerado nesta última etapa é enviado individualmente para cada aluno de maneira automática.

3.2.2 Análise Estática

A Figura 3.12 mostra as etapas de execução da análise estática, as quais serão detalhadas no decorrer da seção.

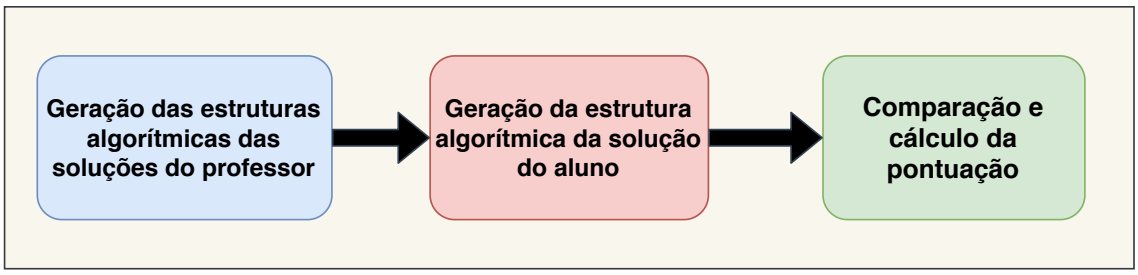

Figura 3.12 – Fluxo de execução da análise estática

Na análise estática, apenas o código do aluno é analisado. Logo, ele é avaliado em termos de complexidade e de sequenciamento correto dos comandos. O objetivo é valorizar a construção da solução e não somente a saída do programa. Para tanto, são definidos arquivos contendo a estrutura algorítmica de possíveis soluções para um problema e essas estruturas são comparadas com a estrutura construída pelo aluno. Para a construção da estrutura algorítmica, tanto da solução do aluno quanto das soluções do professor, são considerados três estruturas da linguagem, sendo elas:

- Laços de repetição
- Estruturas condicionais;
- Funções;

Desconsiderou-se as demais estruturas da linguagem como entrada/saída de dados, comandos de atribuição, declaração de variáveis, dentre outras. Isso porque existem várias formas possíveis de se compor a solução para um problema e, para estas estruturas em específico, a ordem não é relativamente importante, no sentido de que, existem inúmeras formas de se sequenciar os comandos e ainda assim, todas obterem o mesmo resultado. Considerou-se apenas as estruturas acima citadas por serem mais relevantes na construção de uma solução e também por ser inviável gerar todas as possibilidades se considerássemos todas as estruturas.

A análise estática é feita através um pequeno programa que percorre os arquivos de soluções dos alunos e constrói a estrutura algorítmica dos referidos arquivos. Essas estruturas são identificadas através da presença de palavras reservadas da linguagem para os comandos citados anteriormente. Após isso, os arquivos contendo as estruturas algorítmicas da soluções do professor são comparadas com os arquivos que contém a estrutura algorítmica do aluno e tem-se um acúmulo de pontuação na medida em que as soluções comparadas se assemelhem.

Enunciado

Implemente um programa para calcular a média aritmética de números definidos pelo usuário. O usuário deverá primeiramente informar a quantidade de números, e em seguida ler cada número. Então o programa deve calcular e exibir a média aritmética dos números. Perceba que o usuário deverá ser forçado a definir um valor válido para a quantidade de números.

SOLUÇÃO DO PROFESSOR	SOLUÇÃO DO ALUNO

n = *input*("Quantidade de números: ")
 	
while n<=1 | n<>int(n)
  printf ("Quantidade de números inválida!")
  n = *input*("Quantidade de números: ")
end
s = 0
cont = 0
while cont < n
  num= *input*("Número: ")
  s=s+num
  cont=cont+1
end
med= s/n
printf("Média Aritmética: %.2f", med)
	n=*input*("quantidade de números:")
	
	while (n<=1)
	  printf ("Entrada invalida!")
	  n= input("quantidade de números:")
	end
	s=0
	c=0
	while (c<n)
	  k= *input* ("Número:")
	  c=c+1
	  s=s+k
	end
	media=(s/n)
	printf("Média Aritmética:%.2f",media)
	end

ESTRUTURA ALGORÍTMICA DA SOLUÇÃO DO PROFESSOR	ESTRUTURA ALGORÍTMICA DA SOLUÇÃO DO ALUNO
LAÇO DE REPETIÇÃO LAÇO DE REPETIÇÃO	LAÇO DE REPETIÇÃO LAÇO DE REPETIÇÃO

RESULTADO GERADO PELA ANÁLISE DINÂMICA

MENSAGEM DE ERRO: "*Error: syntax error, unexpected end, expecting end of file.*"
MENSAGEM NO PDF: Erro de sintaxe

AVALIAÇÃO

NOTA AVALIAÇÃO DINÂMICA:	0,0 PONTOS	NOTA AVALIAÇÃO ESTÁTICA:	10,0 PONTOS

Figura 3.13 – Avaliação Dinâmica vs Avaliação Estática

Na Figura 3.13 temos um exemplo onde, sob o ponto de vista da análise estática, a solução do aluno é equivalente à solução do professor. Entretanto, devido a um erro sintático, causado pela inclusão de um *end* na última linha do programa, a nota do aluno foi rigidamente penalizada durante a análise dinâmica. Por esse motivo, é necessário não avaliar somente a saída gerada pela solução do aluno mas também o quão esta se aproxima de uma solução ideal. O objetivo é tentar

tornar a correção automática o mais próximo possível de uma correção feita manualmente pelo professor, quando situações deste tipo podem ser facilmente tratadas.

Apesar das abordagens acima citadas, avaliar a estrutura algorítmica do aluno não é uma tarefa trivial. Isso se deve principalmente ao fato de que existem diversas formas de se construir uma solução para um mesmo problema. Além disso, podem existir situações em que na análise estática a estrutura algorítmica do aluno está idêntica a uma das soluções ideais, entretanto a lógica desenvolvida pelo aluno através das estruturas não consideradas está completamente errada. Algumas dessas discussões serão melhores detalhadas na seção de Resultados.

4 Resultados

A proposta metodológica aqui apresentada foi aplicada durante dois semestres letivos para várias turmas de Engenharia da Universidade Federal de Ouro Preto (UFOP). Para melhor visualização dos resultados, separou-se a análise dinâmica da análise estática e por fim, um seção de discussões.

4.1 Análises da Avaliação Dinâmica

Conforme já mencionado, os alunos realizam duas provas teóricas durante o semestre. Assim, as subseções a seguir descreverão uma análise das atividades realizadas para a prova 1, seguida de uma análise das atividades realizadas para a prova 2. Serão apresentados também os resultados obtidos com a aplicação de um questionário aos alunos.

4.1.1 Análises da prova 1

Ao todo, 12 alunos dos 68 envolvidos foram desconsiderados das análises por não serem frequentes e/ou por não realizarem a prova teórica. Para efeito avaliativo, relacionou-se as seguintes análises:

- **Questões objetivas vs. Exercícios práticos:**

  A Figura 4.1 é um gráfico que relaciona a nota média de cada aluno nas questões objetivas com a nota média nos exercícios práticos realizados para a prova 1. Os alunos foram ordenados pela sua nota nas questões objetivas. As linhas de tendência denotam que alunos que resolvem corretamente as questões objetivas tendem a se saírem bem também na resolução dos exercícios práticos.

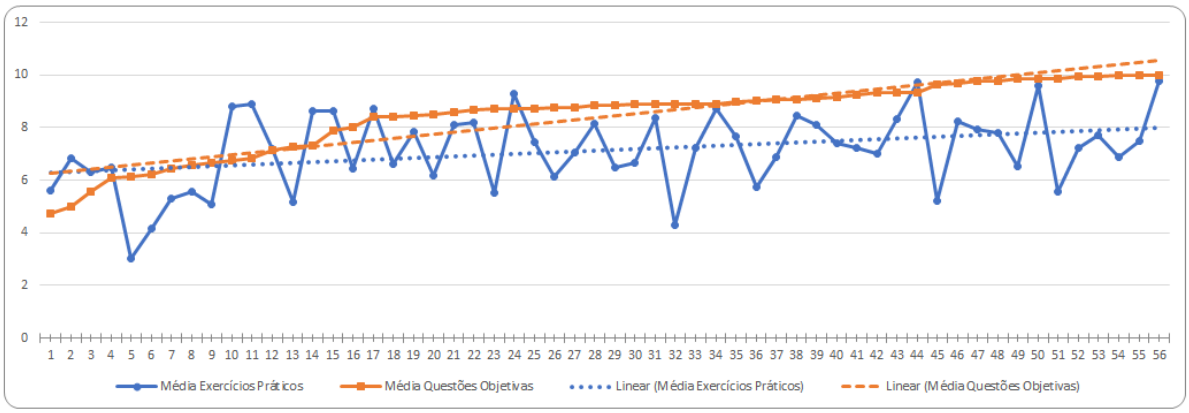

Figura 4.1 – Gráfico comparativo entre questões objetivas e exercícios práticos.

- **Prova teórica vs. Exercícios práticos:**

  Na Figura 4.2 temos um gráfico que compara o desempenho dos alunos na prova teórica e nos exercícios práticos realizados para a prova 1. Os alunos foram ordenados pela sua nota na prova teórica. Novamente temos linhas de tendência crescentes, indicando que alunos que se saem bem na resolução dos exercícios práticos tendem a ter também, um bom desempenho nas avaliações teóricas.

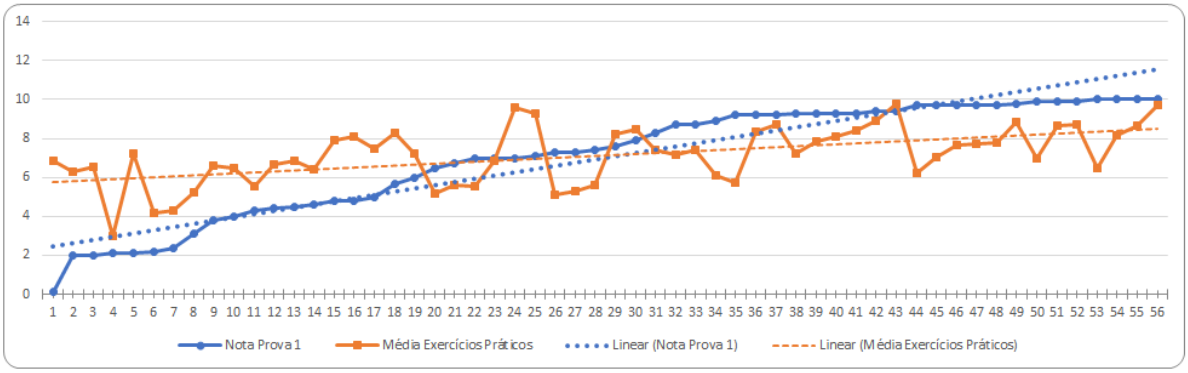

Figura 4.2 – Gráfico comparativo entre prova teórica e exercícios práticos.

4.1.2 Análises da prova 2

Nesta segunda etapa de avaliação, 20 alunos dos 68 envolvidos foram desconsiderados das avaliações por não serem frequentes e/ou por não realizarem a prova teórica. Para efeito avaliativo, relacionou-se as seguintes avaliações:

- **Questões objetivas vs. Exercícios práticos:**

  A Figura 4.3 apresenta um gráfico que compara a nota média de cada aluno nas questões objetivas com a nota média nos exercícios práticos realizados para a prova 2. Os alunos foram ordenados pela sua nota nas questões objetivas. As linhas de tendência crescentes apontaram, mais uma vez, uma tendência de que alunos que obtêm um bom resultado nas avaliações objetivas possuem também bons resultados nos exercícios práticos.

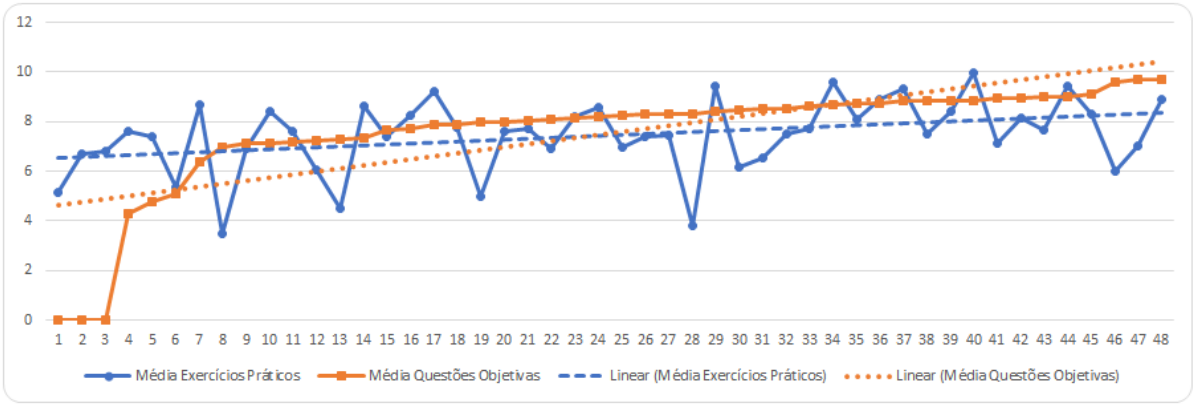

Figura 4.3 – Gráfico comparativo entre os exercícios práticos e as questões objetivas

- **Prova teórica vs. Exercícios práticos:**

  Na Figura 4.4 temos um gráfico que compara a notas dos alunos nos exercícios práticos e suas respectivas notas na segunda prova teórica realizados para a prova 2. Os alunos foram ordenados pela sua nota na prova teórica. Novamente, temos linhas de tendência crescentes, indicando que alunos que se saem bem na resolução dos exercícios práticos tendem a ter também, um bom desempenho nas avaliações teóricas.

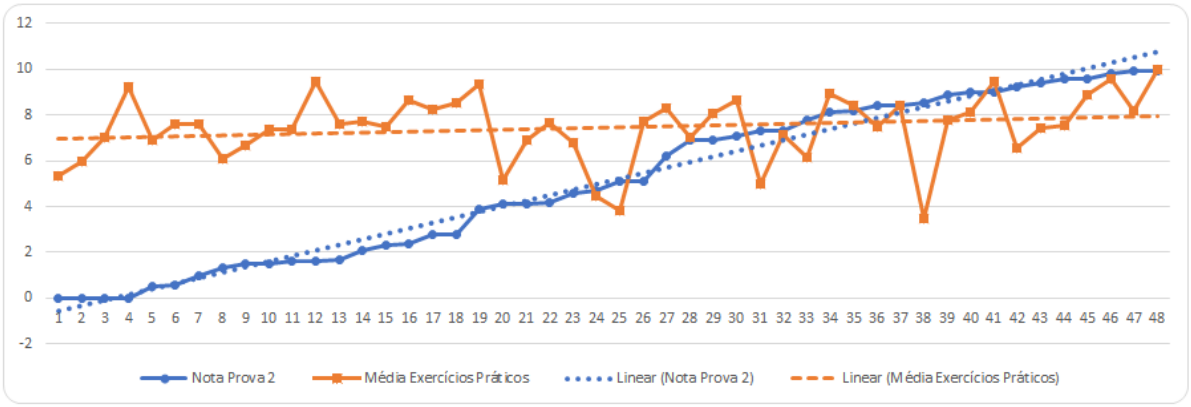

Figura 4.4 – Gráfico comparativo entre prova teórica e exercícios práticos

4.1.3 Análise do questionário de opinião

O questionário foi aplicado para avaliar, sob o ponto de vista do aluno, como o uso da metologia proposta impactou no desempenho acadêmico do mesmo. Ao todo 31 (trinta e um) alunos responderam ao questionário. Exceto a pergunta número 8, que relacionada ao sistema de bonificação/penalização nas entregas dos exercícios práticos, todas as demais foram aplicadas igualmente em ambos os semestres. Para melhor visualização, separou-se esta das demais.

4.1.3.1 Perguntas aplicadas igualmente em ambos os semestres

As perguntas a seguir foram igualmente aplicadas em ambos os semestres. Agrupou-se os dados em um único gráfico para mesma pergunta.

- **Pergunta 1:** Qual foi sua nota final na disciplina (sem considerar exame especial)?

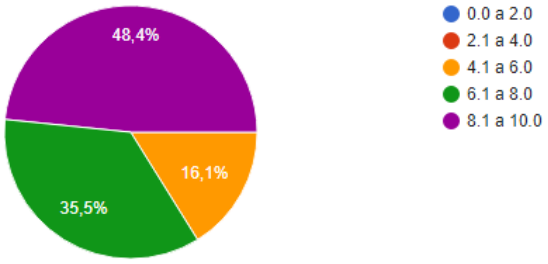

Figura 4.5 – Nota média final

- **Pergunta 2:** Qual foi sua nota média nas avaliações teóricas?

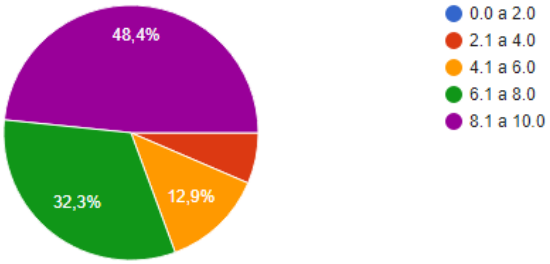

Figura 4.6 – Nota média nas avaliações teóricas

- **Pergunta 3:** Qual foi sua nota média obtida nas avaliações práticas?

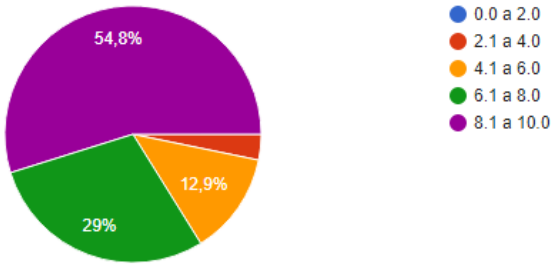

Figura 4.7 – Nota média nas avaliações práticas

- **Pergunta 4:** Qual foi o seu percentual de presença nas aulas?

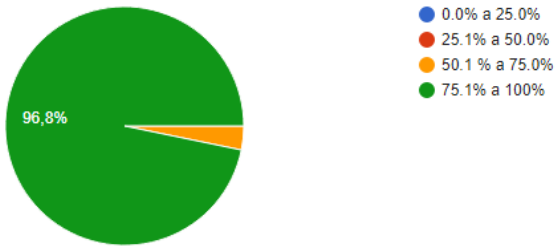

Figura 4.8 – Frequência nas aulas

- **Pergunta 5:** As questões objetivas ajudaram na fixação do conteúdo teórico (1- Discordo Totalmente; 5 - Concordo Plenamente).

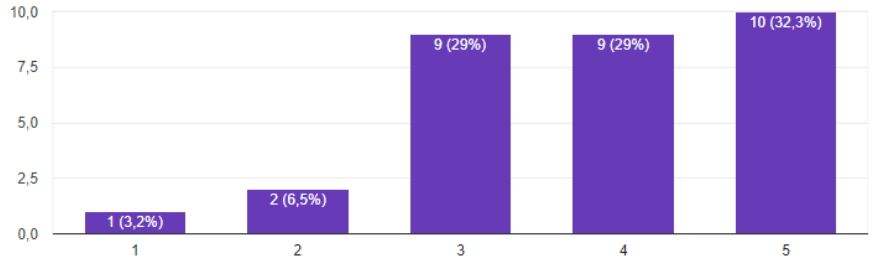

Figura 4.9 – Importância das questões objetivas

- **Pergunta 6:** As questões práticas de programação ajudaram na fixação do conteúdo teórico (1- Discordo Totalmente; 5 - Concordo Plenamente).

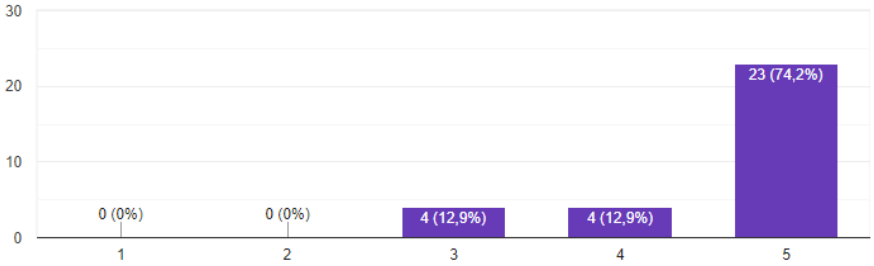

Figura 4.10 – Importância dos exercícios práticos

- **Pergunta 7:** Como você avalia o resultado gerado pelo corretor automático nos exercícios práticos (1-Muito ruim, 5 -Muito bom)?

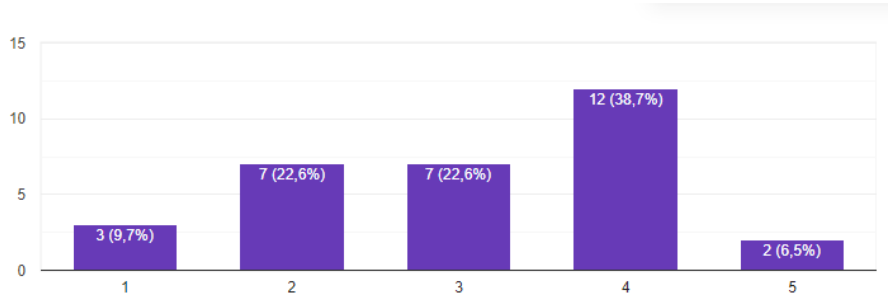

Figura 4.11 – Avaliação do corretor

- **Pergunta 8:** Você já cursou essa disciplina alguma outra vez?

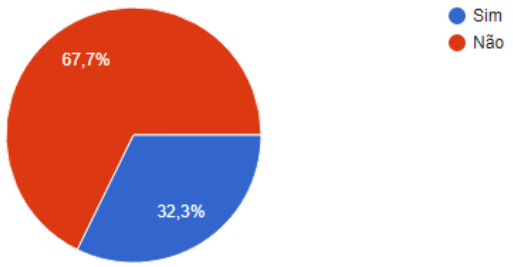

Figura 4.12 – Alunos novatos e repetentes

- **Pergunta 9:** Você já possuía algum conhecimento prévio em programação antes de cursar a disciplina?

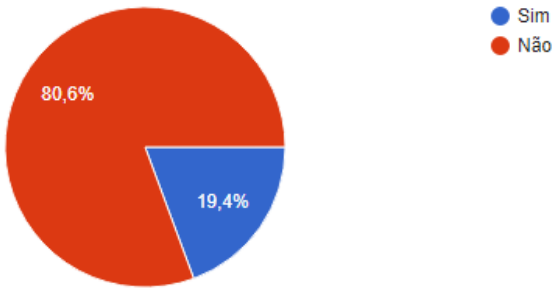

Figura 4.13 – Conhecimento em programação

- **Pergunta 10:** Você considera a disciplina de Programação de Computadores importante para sua formação enquanto engenheiro(a)?

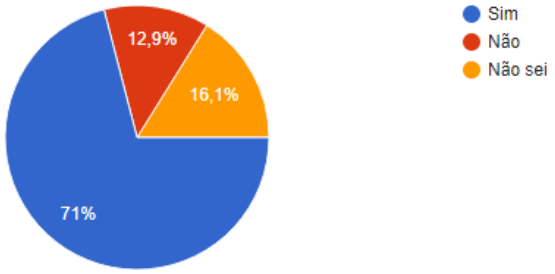

Figura 4.14 – Importância da programação

- **Pergunta 11:** O que mais te ajudou na disciplina?

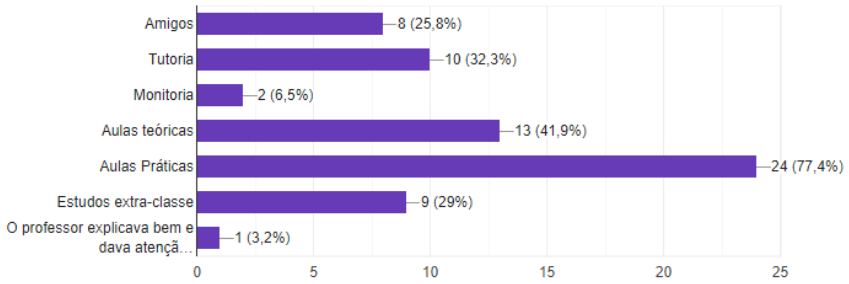

Figura 4.15 – Fatores importantes para o bom desempenho na disciplina

A maior parte dos alunos que responderam ao questionário obtiveram um bom desempenho na disciplina, tanto nas atividades práticas quando na resolução das provas teóricas. Além disso, eram alunos frequentes e consideravam a aplicação de questões objetivas e exercícios práticos como importantes para a fixação do conteúdo teórico.

A avaliação do resultado gerado pelo corretor automático dividiu opiniões. Alguns o consideram uma boa ferramenta de apoio ao processo de aprendizagem, enquanto outros consideram o resultado gerado pelo mesmo ruim. Uma das hipóteses para as avaliações negativas se deve ao fato da ferramenta trabalhar “offline". Ou seja, só depois de um tempo os alunos obtêm a resposta se suas implementações estavam corretas ou não.

A maioria dos alunos que responderam o questionário eram novatos e não possuíam nenhum conhecimento prévio em programação. Além disso, a grande parte considera a disciplina de Programação de Computadores I como importante para sua formação enquanto engenheiro.

Os alunos consideram as aulas práticas como uma das principais razões para se obter sucesso na disciplina, seguida das aulas teóricas e tutoria. Esse fato dá indícios de que a me-

todologia proposta cumpre seu objetivo ao auxiliar o aluno no desenvolvimento do raciocínio lógico.

4.1.3.2 Perguntas relacionadas ao sistema de penalização vs bonificação

As Figuras 4.16 e 4.17 apresentam, respectivamente, a avaliação do aluno sobre o sistema de penalização e o sistema de bonificação. Vale ressaltar que o sistema de penalização foi avaliado por alunos que cursaram a disciplina no semestre anterior enquanto o sistema de bonificação foi avaliado por alunos que cursaram a disciplina neste semestre.

- **Pergunta 12-A:** As penalidades por ausência e horário motivaram você a realizar as atividades práticas durante as aulas no laboratório (1- Não ajudaram em nada, 5 - Ajudaram bastante)?

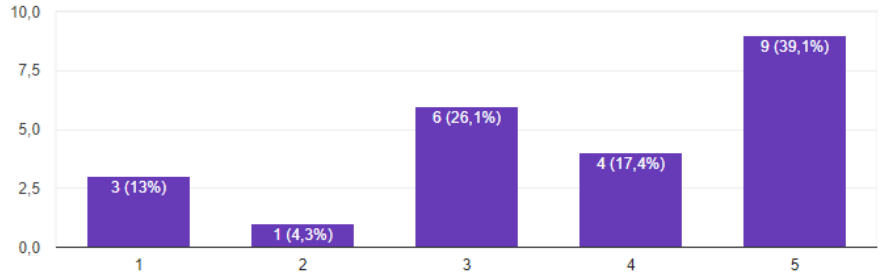

Figura 4.16 – Avaliação do uso das penalidades

- **Pergunta 12-B:** As As bonificações na nota motivaram você a realizar as atividades práticas durante as aulas no laboratório? (1- Não ajudaram em nada, 5 - Ajudaram bastante)?

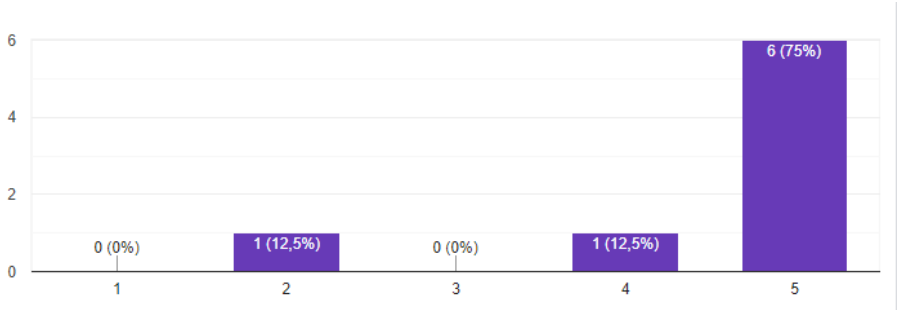

Figura 4.17 – Avaliação do uso das bonificações

Conforme dito, no primeiro semestre utilizamos as penalizações por ausência e horário como estratégia para incentivar os alunos a realizarem suas atividades na sala de aula. Em geral, a proposta obteve sucesso, mas não agradou a todos. Nesse segundo semestre, visando explorar novas estratégias, optou-se por utilizar um sistema de bonificação como forma de motivação aos alunos. Comparado ao semestre anterior, neste semestre uma maior porcentagem dos alunos

responderam que a pontuação bônus os motivaram a realizar as atividades práticas durante as aulas. Entretanto, no semestre anterior um maior número de alunos responderam ao questionário.

4.2 Análises da Avaliação Estática

Conforme dito anteriormente, sob o ponto de vista pedagógico, realizar apenas a avaliação dinâmica da solução do aluno pode desmotivá-lo em seu processo de aprendizagem. Isso porque pequenos erros sintáticos, ou mesmo lógicos, trazem severas penalizações na nota. Realizou-se um estudo experimental avaliando 207 exercícios entregues pelos alunos. Para cada exercício, identificou-se aqueles que possuíam alta discrepância na nota obtida pela análise estática e dinâmica. Identificados estes casos, realizou-se um trabalho manual no intuito de verificar o motivo de tais ocorrências. Dentre os 207 exercícios avaliados, 32 possuíam alta pontuação na análise estática e pontuação mínima na avaliação dinâmica. Analisando código a código constatou-se que essa discrepância foi causada por três principais erros. A Figura 4.18 apresenta estes erros e a porcentagem em que eles apareceram.

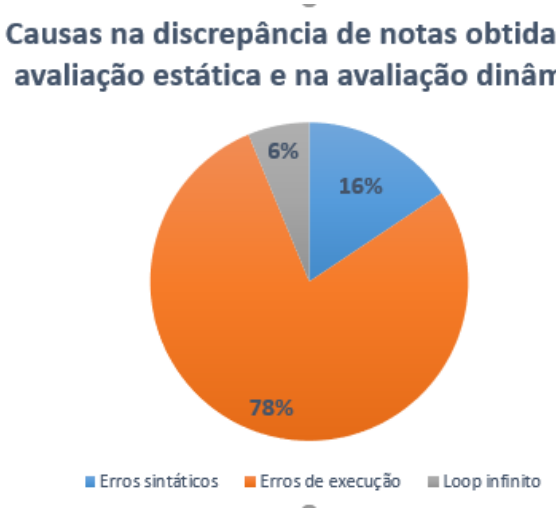

Figura 4.18 – Erros que causaram discrepância entre as notas obtidas na avaliação estática e dinâmica

Identificados tais erros e avaliando o que seria mais próximo de uma correção feita manualmente pelo professor, pode-se levantar as seguintes observações:

- Exceto quando o uso de funções é obrigatório, não é necessário realizar a análise estática quando o aluno obtém uma boa pontuação na análise dinâmica. Isso porque os casos de testes da análise dinâmica simulam as mais diferentes possibilidades de entrada e saída para o programa em questão. Logo, se as saídas obtidas na execução da solução do aluno são equivalentes às saídas esperadas, não é necessário realizar análise do código.

- Se, a partir da análise dinâmica o aluno obteve nota zero em uma questão e essa pontuação foi atribuída devido a um erro sintático e, se na análise estática ele obteve uma boa pontuação, deve-se ponderar a nota atribuindo um maior peso para a análise estática.

- O loop infinito é um erro bastante comum durante o aprendizado de lógica de programação. Quando esse tipo de erro acontece, o corretor automático não obtém uma saída, logo é atribuída nota zero à solução do aluno pela análise dinâmica. Novamente, neste caso deve-se ponderar a nota atribuindo um maior peso para a análise estática, visto que potencialmente o erro foi causado pelo não controle correto de uma determinada *flag*.

4.3 Discussões

Com relação à análise dinâmica algumas questões puderam ser observadas. Como justificar, por exemplo, alunos que obtêm um bom desempenho nas atividades práticas entretanto vão muito mal nas avaliações teóricas. Algumas hipóteses foram levantas no intuito de realizar um estudo mais aprofundado em trabalhos futuros para estes casos, são elas:

- **H1:** Copiou os exercícios práticos do colega sem ao menos ler o que estava sendo entregue.
- **H2:** Consegue realizar as atividades no computador mas no papel não.
- **H3:** Fatores psicológicos que afetaram na concentração da resolução da prova teórica.
- **H4:** Prazo maior para resolver as questões práticas.
- **H5:** Auxílio do monitor, tutor e professor nas aulas práticas, algo que não ocorre durante as provas.

Com relação às análises dinâmica e estática podemos observar que, do ponto de vista pedagógico, aplicar apenas uma delas não é interessante, visto que podem penalizar consideravelmente um aluno que desenvolveu um raciocínio lógico próximo do que era esperado pelo professor. Uma estratégia é gerar o resultado para o aluno a partir de ponderações dos resultados obtidos pela análise estática e dinâmica que mais se aproxime de uma avaliação manual realizada pelo professor.

Por fim, especificamente em relação à análise estática, observou-se que estruturas mais complexas necessitariam de uma análise mais aprofundada para gerar uma avaliação mais concisa. Isso porque, conforme já foi dito, existem inúmeras formas de se resolver um mesmo problema e por isso, afirmar com exatidão a corretude de uma solução torna-se uma tarefa não trivial.

5 Considerações Finais

A proposta metodológica aqui descrita baseou-se no uso de um corretor automático e da exploração de questões objetivas para auxiliar o processo ensino-aprendizagem de alunos da Engenharia na Disciplinas de Programação de Computadores I. A metodologia trouxe importantes resultados, sobretudo no que diz respeito ao rendimento acadêmico dos alunos que a seguiram. Foi possível constatar que alunos que realizam as atividades práticas tendem a se sair bem nas avaliações teóricas. Além disso, sob o ponto de vista do aluno, as aulas práticas foram muito importantes para se obter sucesso na disciplina.

5.1 Trabalhos Futuros

Como trabalhos futuros propõe-se:

1. Responder às hipóteses levantadas durante análise dos resultados obtidos;
2. Explorar diferentes pesos para análises dinâmica e estática para compor a nota final;
3. Explorar conceitos de linguagem formal para aprimorar a análise estática;
4. Reaplicar a metodologia sem utilizar o sistema de bonificação e/ou penalização;
5. Expandir a ferramenta desenvolvida para trabalhar de maneira online.

5.2 Publicações Realizadas

Publicou-se um artigo de QUALIS B1 referente a este trabalho no Simpósio Brasileiro de Informática na Educação 2019, realizado em Brasília:

- BRITO, Palloma; FORTES, Reinaldo. O uso de corretores automáticos para o ensino de programação de computadores para alunos de engenharia. Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE), [S.l.], p. 449, nov. 2019. ISSN 2316-6533. Disponível em: <https://www.br-ie.org/pub/index.php/sbie/article/view/8749>>. doi: <http://dx.doi.org/10.5753/cbie.sbie.2019.449>.

Referências

aaaa.

Berssanette, João Henrique de Francisco, & Carlos, Antonio. 2018. Uma Proposta de Ensino de Programação de Computadores com base na PBL utilizando o portal URI Online Judge. *Anais do Simpósio Ibero-Americano de Tecnologias Educacionais*, 348–354.

Carreño-León, M., Sandoval-Bringas, A., Álvarez Rodríguez, F., & Camacho-González, Y. 2018 (April). *Gamification technique for teaching programming*.

Gonçalves, Dimas Antônio Silveira, da Silva, Gislaine Moura, da Luz, Ronaldo Santos, & Silva, Eduardo Paulino. 2013. Relato de experiência de alunos do curso de Licenciatura em Computação do IFMG-campus Ouro Branco na utilização de objetos de aprendizagem desplugados e do Scratch como instrumentos no ensino de programação. *In: Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, vol. 2.

Jesus, G., Santos, K., Conceicao, J., Ribeiro, E., & Neto, A. 2018. Avaliação de uma abordagem para auxiliar a correção de erros de aprendizes de programação. BRAZILIAN SYMPOSIUM ON COMPUTERS IN EDUCATION, 2018, Fortaleza. Anais do SBIE 2018 (Proceedings of the SBIE 2018).

Levenshtein, Vladimir Iosifovich. 1966. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady*, 10(8), 707–710. Doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

Marcussi, Leticia D, Guedes, Karoline, Dal Molin Filho, Rafael Germano, Santiago Filho, Robertino Mendes, & Junior, Carlos Roberto Beleti. 2016. PESQUISA NO ENSINO DE ALGORITMOS E PROGRAMAÇÃO NAS ENGENHARIAS: ESTUDOS E RESULTADOS PRELIMINARES. *In: Simpósio de Engenharia de Produção*.

Piekarski, Ana Elisa, Miazaki, Mauro, Hild, Tony, Mulati, Mauro Henrique, & Kikuti, Daniel. 2015. *A metodologia das maratonas de programação em um projeto de extensão: um relato de experiência*.

Raabe, André, de Jesus, Elieser Ademir, Hodecker, Andrei, & Pelz, Fillipi. 2015. Avaliaçao do feedback gerado por um corretor automático de algoritmos. *Page 358 of: Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*, vol. 26.

Sgoti, Rogerio Ferreira, & Mill, Daniel. 2018. GAMIFICAÇÃO DE UMA LISTA DE EXERCÍCIOS DA DISCIPLINA ALGORITMOS E ANÁLISE DE SUA APLICAÇÃO EM SUBSTITUIÇÃO AO MODO CONVENCIONAL. *CIET: EnPED*.

Silva, Tatyane, Melo, Jeane, & Tedesco, Patrícia. 2016. *Um modelo para promover o engajamento estudantil no aprendizado de programação utilizando gamification*.

Apêndices

APÊNDICE A – Exemplo de um PDF gerado pelo corretor

[18/09] Estrutura de Repetição (prática 2)

NOME DO ALUNO

Questão Teórica	Nota
1	10.0
2	10.0
3	10.0
Extra	1 .0
Nota Exercícios Teóricos	**11.0**

Questão Prática	Nota
4	10.0
5	10.0
6	10.0
Extra	1 .0
Nota Exercícios Práticos	**11.0**
Nota Final (0.3 * 11.0 + 0.7 * 11.0)	**11.0**

Observações:

- Questões teóricas são corrigidas pelo próprio Moodle.
- Questões práticas são corrigidas pelo nosso corretor, os detalhes da correção são apresentados nas páginas a seguir.
- Ponto extra é computado para entregas feitas antes do encerramento da aula prática.

Questão 4

Caso de teste	Nota
1	10.0
2	10.0
3	10.0
4	10.0
Nota Final	**10.0**

Caso de teste 1

```
Quantidade de termos: -10
Quantidade de termos: 0
Quantidade de termos: 1
Somatório: 0.5
```

Saída esperada	Saída obtida	Pontuação
Somatório: 0.5	Somatório: 0.5	10.0

Caso de teste 2

```
Quantidade de termos: 4
Somatório: 1.04167
```

Saída esperada	Saída obtida	Pontuação
Somatório: 1.04167	Somatório: 1.04167	10.0

Caso de teste 3

```
Quantidade de termos: 50
Somatório: 2.2496
```

Saída esperada	Saída obtida	Pontuação
Somatório: 2.2496	Somatório: 2.2496	10.0

Caso de teste 4

```
Quantidade de termos: 100
Somatório: 2.59369
```

Saída esperada	Saída obtida	Pontuação
Somatório: 2.59369	Somatório: 2.59369	10.0

Questão 5

Caso de teste	Nota
1	10.0
2	10.0
3	10.0
4	10.0
Nota Final	**10.0**

Caso de teste 1

```
Quantidade de termos: -10
Quantidade de termos: 0
Quantidade de termos: 1
Somatório: 1
```

Saída esperada	Saída obtida	Pontuação
Somatório: 1	Somatório: 1	10.0

Caso de teste 2

```
Quantidade de termos: 4
Somatório: 6.41667
```

Saída esperada	Saída obtida	Pontuação
Somatório: 6.41667	Somatório: 6.41667	10.0

Caso de teste 3

```
Quantidade de termos: 35
Somatório: 114.284
```

Saída esperada	Saída obtida	Pontuação
Somatório: 114.284	Somatório: 114.284	10.0

Caso de teste 4

```
Quantidade de termos: 100
Somatório: 423.925
```

Saída esperada	Saída obtida	Pontuação
Somatório: 423.925	Somatório: 423.925	10.0

Questão 6

Caso de teste	Nota
1	10.0
2	10.0
3	10.0
4	10.0
Nota Final	**10.0**

Caso de teste 1

```
Quantidade de termos: -10
Quantidade de termos: 0
Quantidade de termos: 1
Valor aproximado de pi: 2.6667
```

Saída esperada	Saída obtida	Pontuação
Valor aproximado de pi: 2.6667	Valor aproximado de pi: 2.6667	10.0

Caso de teste 2

```
Quantidade de termos: 0.567
Quantidade de termos: 5
Valor aproximado de pi: 3.0418
```

Saída esperada	Saída obtida	Pontuação
Valor aproximado de pi: 3.0418	Valor aproximado de pi: 3.0418	10.0

Caso de teste 3

```
Quantidade de termos: 1000
Valor aproximado de pi: 3.1411
```

Saída esperada	Saída obtida	Pontuação
Valor aproximado de pi: 3.1411	Valor aproximado de pi: 3.1411	10.0

Caso de teste 4

```
Quantidade de termos: 10000
Valor aproximado de pi: 3.1415
```

Saída esperada	Saída obtida	Pontuação
Valor aproximado de pi: 3.1415	Valor aproximado de pi: 3.1415	10.0