UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL GOMES SOUSA

MÉTODO DE COMPOSIÇÃO DE RESULTADO PARA RESOLUÇÃO DE PROBLEMAS: MELHORIA NO DESEMPENHO DA APRENDIZAGEM DO ESTUDANTE EM ATIVIDADES PRÁTICAS DE PROGRAMAÇÃO

Belém
2016

MÉTODO DE COMPOSIÇÃO DE RESULTADO PARA RESOLUÇÃO DE PROBLEMAS: MELHORIA NO DESEMPENHO DA APRENDIZAGEM DO ESTUDANTE EM ATIVIDADES PRÁTICAS DE PROGRAMAÇÃO

Rafael Gomes Sousa

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Ciência da Computação, PPGCC, da Universidade Federal do Pará, como parte dos requisitos necessários à obtenção do título de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Eloi Luiz Favero

Belém
2016

Dados Internacionais de Catalogação - na - Publicação (CIP)
Biblioteca de Pós-Graduação do ICEN/UFPA

Sousa, Rafael Gomes
Método de composição de resultado para resolução de problemas: melhoria no desempenho da aprendizagem do estudante em atividades práticas de programação /Rafael Gomes Sousa; orientador, Eloi Luiz Favero.-2016.
115 f.: il; 29 cm
Inclui bibliografias

Dissertação (Mestrado) – Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais, Programa de Pós-Graduação em Ciência da Computação, Belém, 2016.

1. Programação (computadores)- Estudo e ensino. 2. Programação (computadores)-Conhecimentos e aprendizagem. 3. Estratégias de aprendizagem – Programação-Metodologia. 4. Método de resolução de problemas-Programação. 5. Ensino-Metodologia- Programação. I. Título.

CDD – 22 ed. 005.1071

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL GOMES SOUSA

MÉTODO DE COMPOSIÇÃO DE RESULTADO PARA RESOLUÇÃO DE PROBLEMAS: MELHORIA NO DESEMPENHO DA APRENDIZAGEM DO ESTUDANTE EM ATIVIDADES PRÁTICAS DE PROGRAMAÇÃO

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Pará como requisito para obtenção do título de Mestre em Ciência da Computação, defendida e aprovada em 29/12/2016, pela banca examinadora constituída pelos seguintes membros:

Prof. Dr. Elói Luiz Favero
Orientador – PPGCC/UFPA

Prof. Dr. Bianchi Serique Meiguins
Membro Interno – PPGCC/UFPA

Prof. Dr. João Carlos Alves dos Santos
Membro Externo – FAMAT/UFPA

Visto:
Prof. Dr. Jefferson Magalhães de Morais
Coordenador do PPGCC/UFPA

Prof. Dr. Jefferson Magalhães de Morais
Coordenador do PPGCC
Mat.: SIAPE: 2378314

Dedico este trabalho a todos que de alguma forma contribuíram para o seu desenvolvimento, em especial para meu sobrinhos filhos: Igor e Iago.

Agradecimentos

Agradeço primeiramente a Deus por me oferecer caminhos que me fizeram chegar onde estou e para rumo que seguirei. Agradeço especialmente a pessoa que me ajudou imensamente nessa jornada, meu orientador, professor doutor Eloi Favero.

Ao meu grande amigo Bruno que me deu muito incentivo e apoio nessa caminhada do mestrado, das disciplinas ao fechamento da dissertação.

Aos coordenadores do programa de mestrado, professores Nelson e Jefferson pelo enorme apoio e compreensão.

Ao professor Benedito que me abriu as portas do programa de mestrado.

A professora Marianne pela enorme colaboração no estágio em docência, em que pude ter a vivência prática, inicial, com ensino de programação.

Agradeço aos meus familiares e pessoas próximas que me deram apoio para que eu pudesse vencer essa difícil batalha.

O curso do amor verdadeiro nunca fluiu suavemente.
William Shakespeare

Resumo

O processo de ensino e aprendizagem de programação é uma tarefa bastante complexa, para tanto, defendemos a abordagem prática de resolução de problemas para o melhor aproveitamento acadêmico dos estudantes. Entretanto, a construção da resposta para um problema possui uma alta carga de subjetividade. Aqui, concebemos um método resolução de problemas de programação (Composição de Resultado), auxiliado por um ferramental de apoio (Ambiente de ensino LabProg Web). Para avaliar o método e o ferramental fizemos um experimento coletando informações sobre a satisfação dos estudantes e professores que participaram. Os resultados mostraram melhorias expressivas na atuação de estudantes que fizeram uso dos produtos de trabalho dessa dissertação, em atividades práticas de resolução de problemas de programação.

Palavras-Chaves: Treino de programação, ensino de programação, correção automatizada, método de resolução de problemas.

Abstract

The process of teaching and learning programming is a very complex task, so we defend the practical approach of problem solving for the best academic achievement of students. However, the construction of the response to a problem has a high subjectivity load. Here, we conceive a method of solving programming problems (Result Composition), aided by a support tooling (LabProg Web Teaching Environment). To evaluate the method and the tooling we did an experiment collecting information about the satisfaction of the students and teachers who participated. The results showed significant improvements in the performance of students who made use of the work products of this dissertation, in practical activities to solve programming problems.

Keywords: programming training, educational programming, automated remediation, problem-solving method.

Lista de Figuras

Lista de Tabelas

Sumário

Capítulo 1

Introdução

Este capítulo apresenta a introdução do trabalho, descrevendo as motivações, os objetivos, a metodologia e pôr fim a estrutura da dissertação.

1.1 Problemática

O ensino de programação é um item, dos mais importantes, na grade curricular dos cursos de computação e áreas afins, sendo sempre pré-requisito para várias outras disciplinas. Assim, o ensino de programação interfere tanto no direcionamento quanto na formação de profissionais da computação. Logo, o insucesso nesse item pode acarretar diversas consequências indesejadas durante à jornada dos estudantes, pois exige a compreensão correta de conceitos abstratos, fazendo com que muitos estudantes tenham problemas no processo de aprendizagem como destacado por Lahtinen, Ala-Mutka e Jarvinen (2005).

Segundo Proulx (2000) a dificuldade no ensino de programação se transforma em uma grande barreira, acarretando um grau elevado de reprovações e desistência em disciplinas de formação, que em muitos casos, ocasiona evasões em cursos de computação. Destacando os motivos da dificuldade (PROULX, 2000):

- Preocupação excessiva com detalhes de sintaxe da linguagem usada (C, Pascal, Java, etc.);
- Falta de visão para idealizar soluções que quando mapeadas produzam passos sequenciais e de abstração do funcionamento dos mecanismos escolhidos;
- Estabelecimento de um raciocínio lógico visando à resolução de problemas, com base em um modelo incremental, em relação à complexidade e à estratégia de refinamentos sucessivos.

De acordo com Scaico et al. (2012), aprender a programar é extremamente importante, principalmente, ao se considerar que o desenvolvimento de algoritmos é o eixo central para todas as áreas relacionadas a Computação. Contudo, esta tarefa requer o domínio

de um conjunto amplo de habilidades técnicas, tais como o conhecimento de linguagens de programação, de ambientes para a construção do código, de ter um embasamento matemático, e de outras capacidades que estão mais relacionadas a aspectos cognitivos e psicológicos. Como exemplos dessas capacidades, podem ser citadas as habilidades de representar bem o problema e saber decompô-lo, de construir o seu significado, de criar modelos mentais que auxiliem a busca da solução, de abstrair conceitos e de reconhecer padrões (SUDOL, 2011).

Na abordagem tradicional, os fundamentos de programação são geralmente ensinados utilizando uma linguagem de programação específicas (HICKEY, 2004). Por exemplo, estudantes encontram dificuldades quando a ênfase no ensino é focada na sintaxe, em vez da resolução de problemas através do desenvolvimento de algoritmos. Desta forma, estudantes aprendem a programar por meio de uma prática de tentativa e erro, sem desenvolver as habilidades de compreensão e análise (EDWARDS, 2004).

Os alunos necessitam desenvolver competências para conceber programas capazes de resolver problemas, no entanto, demonstram grande dificuldade em aplicar conceitos abstratos de programação. Portanto, em aplicações básicas como estruturas de controle e repetição, compreendem os conceitos, porém, possuem dificuldades no uso prático dessas estruturas. O trabalho de Gomes e Mendes (2007) aponta que a causa mais importante na origem das dificuldades de programação reside em um déficit na capacidades de resolução de problemas genéricos, pois a resolução de problemas requer múltiplas competências; tais como:

- Compreensão - estudantes iniciam a codificação antes de entender o que está sendo solicitado;
- Relacionamento - os aprendizes tendem a relacionar a solução anteriores, ou seja, quando encontram um raciocínio diferente tendem a implementar solução errada;
- Reflexão - na construção da solução fazem testes superficiais e consideram correto sem fazer testes extensivos;
- Persistência - os alunos tendem a desistir de problemas cujas soluções não conseguem encontrar de forma simples e rápida.

1.2 Motivação

Segundo Aharoni (2000), o processo de ensino e aprendizado de programação é crucial não apenas porque programar é uma "habilidade prática", mas também, porque o processo é o meio de formar conhecimento efetivo e conceitual. Tradicionalmente, a comunidade acadêmica atribuiu e ainda tem atribuído muita importância ao tipo de conhecimento conceitual que os estudantes precisam adquirir e a relação hierárquica daquele conhecimento.

Deste modo, temos uma ênfase pedagógica focada no conteúdo, sendo realizada por professores que adotam ou recorrem a tais práticas deducionais. Corroborando com essa ideia, os trabalhos de Bower (2008) e Aharoni (2000) relatam que o problema do ensino e aprendizado de programação não se encontra no nível de conteúdo, e sim, a nível do processo que define como os conteúdos são ministrados, e quais habilidades são exigidas para dominar o conhecimento nessa área.

Considerando a importância da prática nas atividades de programação, Lahtinen, Ala-Mutka e Jarvinen (2005) afirmam que a dificuldade não ocorre somente por conta dos conceitos abstratos, pois os alunos possuem problemas em diferentes questões relacionadas a construção do programa. Portanto, é importante que pratiquem bastante com materiais e abordagens cuidadosamente projetados, sendo que os professores devem orientar a construção do conhecimento e habilidades.

Para que possam adquirir, melhorar ou mesmo manter uma habilidade complexa, é necessário praticar com regularidade (GROSS; POWERS, 2005), (ECKERDAL, 2009). A quantidade de prática requerida depende da natureza da atividade em cada indivíduo. Nesta perspectiva, pode-se relacionar a melhora da aprendizagem dos indivíduos com suas capacidades inerentes e experiências anteriores sobre o domínio estudado. Sobre este aspecto, o feedback é uma ferramenta que pode proporcionar melhorias no processo de aprendizagem, pois quando inexistente ou inadequado, promove práticas ineficientes ou prejudiciais ao aprendizado.

O trabalho de Lahtinen, Ala-Mutka e Jarvinen (2005) mostra que os alunos preferem aprender através de muita prática com exercícios, a aprender através de aulas expositivas. Dessa forma, é importante que o professor utilize exercícios como forma de estímulo e avaliação dos alunos. Nesse contexto, estudos, como Truong, Roe e Bancroft (2005) e Singh, Gulwani e Solar-Lezama (2013) mostram a importância de ofertar feedback, e assim fornecer aos alunos um auxílio no processo de aprendizagem.

Considerando a necessidade de se trabalhar com exercícios e de fornecer subsídio ao estudante, realiza-se pesquisas no sentido de desenvolver ferramentas que auxiliem o professor nesse processo, como nos trabalhos de Douce, Livingstone e Orwell (2005), Ihantola et al. (2010) e Pieterse (2013). Para que seja possível trabalhar com turmas grandes e proporcionar retorno rápido acerca das atividades desenvolvidas, as ferramentas utilizam alguma forma de avaliação automática.

No contexto das ferramentas de ensino, Santos e Costa (2006) esclarecem que ferramentas automatizadas são importantes, mas devem ser cuidadosamente estudadas para sua implantação como apoio ao ensino. Pois, elas são eficientes apenas se estiverem atreladas e baseadas em um método específico de ensino.

Até então, o que mais se vê é o uso de estratégias isoladas de ensino e ferramentas baseadas em teorias educacionais, como em Borges (2000). Poucos são os trabalhos que se ocuparam da produção de métodos específicos de ensino e aprendizagem atrelados a ambientes de programação.

1.3 Hipótese

Considerando a relevância do ensino e aprendizagem prático de programação de computadores e que ainda é necessário promover muitas melhorias nesse processo, esta dissertação de mestrado testa a seguinte hipótese:

Propor um método de resolução de problemas de programação que oriente a construção do algoritmo, promovendo melhoria no desempenho da aprendizagem do estudante em atividades práticas.

1.4 Objetivo

Conceber um método de resolução de problemas de programação de computadores, buscando melhorar o desempenho da aprendizagem dos estudantes em atividades práticas de programação, assim como oferecer o suporte de infraestrutura lógica e de software necessários a sua execução. Facilitando o trabalho do professor com a solução proposta.

1.4.1 Objetivos Específicos

- Identificar funcionalidade e características para o método e ferramentas de software;
- Propor um método de resolução de problemas de programação;
- Desenvolver ferramentas de suporte ao método;
- Disponibilizar recursos de suporte para as atividades do professor;
- Fornecer mecanismos de correção automatizada para os estudantes;
- Apresentar orientações para condução da resolução;
- Propor indicador para avaliar a resposta do estudante;
- Demonstrar melhoria de desemprenho por meio de experimento;
- Levantar informações a respeito da satisfação de alunos e apreciação de professores.

1.5 Metodologia do Trabalho

Considerando o foco desse trabalho, que visa promover melhorias no ensino e aprendizagem de programação em atividades práticas. Assim, temos como interesse em estudantes que já possuem um contato inicial com programação ou introdução a algoritmos. Pois o foco não é, necessariamente, ensinar as estruturas básicas, e sim empregar esses conhecimentos na resolução de problemas de programação.

A proposta desse trabalho foi validada em duas fases: um experimento prático em sala de aula e um questionamento sobre a satisfação dos principais atores envolvidos (professores e alunos).

No primeiro experimento com estudantes, eles fizeram uso do método e das ferramentas, como forma de analisar a viabilidade de utilização do método da Composição de Resultado. Posteriormente responderam um questionário sobre a satisfação.

No segundo experimento, os produtos desse trabalho foram apresentados para professores que trabalham com ensino de programação. Foi colhida a apreciação desses profissionais acerca do método e das ferramentas.

O primeiro experimento foi realizado com uma turma de 39 alunos do curso de ciência da computação, que cursavam segundo semestre na disciplina de Linguagem de Programação Estruturada, que tem o foco no treino das habilidades práticas de programação. Eles já haviam cursado no semestre anterior a disciplina Lógica de Programação e Algoritmos.

A operacionalização da medição do desempenho educacional dos estudantes foi realizada por meio da análise das médias de acerto dos problemas. Seguida por um teste estatístico que indica se a diferença encontrada é real, ou seja, estatisticamente significante.

A apreciação dos professores foi realizada com sete profissionais que atuavam no ensino de programação com experiência variando de dois a oito anos, e idade predominante na faixa de 31 a 40 anos.

Os elementos de satisfação e apreciação foram analisados por meio de uma abordagem quantitativa, fazendo uso da estatística descritiva que trabalha em um processo de quantização de elementos categóricos, para extração de informações.

1.6 Organização do Texto

Neste capítulo introdutório foram apresentados o problema motivador deste estudo e a hipótese gerada como resposta. Os objetivos para validar a hipótese, a relevância e o escopo da pesquisa deste trabalho também foram descritos. A dissertação prossegue com a seguinte organização:

Capítulo 2. Apresenta o referencial teórico, e trabalhos com abordagens próximas (e) ou importantes para este estudo.

Capítulo 3. Apresenta um caminho para resolução de problemas de programação, formando o método de Composição de resultado. Portanto, o estudante obtêm um mecanismo para iniciar e progredir na resolução de problemas de programação.

Capítulo 4. Descreve do Ambiente de Aprendizagem, detalhando os vários sistemas que o compõe, descrevendo e ilustrando o seu funcionamento. Além de relacionar como a utilização dos softwares auxiliam a execução do método de resolução proposto.

Capítulo 5. Descreve os experimentos em sala de aula com os estudantes de programação e a apreciação dos professores. Apresenta o resultados dos experimentos realizados,

bem como uma discussão da contribuição dele em cada cenários.

Capítulo 6. Apresenta as considerações finais e uma breve discussão dos resultados confrontando com a hipótese e objetivos definidos, além de desdobramentos futuros da pesquisa.

Capítulo 2

Referencial Teórico

Neste capitulo apresentaremos os conceitos para fundamentação deste trabalho, algumas abordagens relacionadas ao modo de promover o ensino de programação, e uma seleção de trabalhos relacionados.

2.1 Algoritmos e Problemas

Um algoritmo é um conjunto de regras que, para um conjunto de entradas, irá produzir uma saída específica (KNUTH, 1968). Portanto, para cada entrada realiza-se o processamento em um número finito de passos, chegando-se um resultado. Os passos podem ser traduzidos em uma linguagem de programação e executado por uma máquina.

2.1.1 Problemas

Uma definição, amplamente aceita, identifica um problema como uma situação que apresenta dificuldades a um indivíduo para as quais não há solução imediata (MAZARIO, 2002). A partindo dessa definição, uma situação só será considerada um problema quando o indivíduo não dispõe de conhecimento e(ou) procedimentos que lhe permitam solucionar o problema de forma imediata. Desta forma é exigido reflexão e tomada de decisão sobre às estratégias serão empregadas para a sua resolução (POZO, 1998).

O trabalho de Mendonça et al. (2010) apresenta dois domínios relacionados, considerando de forma mais profunda a resolução de um problema:

Espaço do problema é a descrição e caracterização do problema a ser resolvido. Assim, o termo está associado à definição "do que" deve ser feito;

Espaço da solução é o conjunto das soluções possíveis para um dado problema. Deste modo, o termo está associado ao "como" o problema será solucionado.

Para Chi, Glaser e Farr (2014), um problema é uma situação na qual o indivíduo atua com o propósito de atingir uma meta, utilizando alguma estratégia em particular. Sendo

comum confundir a palavra problema com exercício, sendo que muitas vezes, eles são utilizados como equivalentes. Entretanto, o exercício envolve mera aplicação de conteúdos ministrados, enquanto o problema necessariamente envolve invenção e/ou criação que pode ser considerada como significativa.

Dentro das possíveis concepções podemos destacar a de Jonassen (2000) relata problemas bem estruturados como sendo os que apresentam no enunciado todas as informações necessárias ao seu entendimento (estado inicial, objetivo e os operadores que transformam o estado do problema). E que nesse contexto requerem a aplicação de um limitado número de conceitos, regras e princípios, envolvendo procedimentos bem especificados e conhecidos. E que ainda podem ser comumente encontrados em trabalhos universitários e nos livros textos.

Já para Hong (1998) problemas mal estruturados, são aqueles cuja descrição é menos clara. As informações disponíveis no seu enunciado são incompletas e ambíguas, podendo até mesmo serem contraditórias. Esses problemas, geralmente, não requerem informações de uma única área de conhecimento, podem possuir múltiplas soluções todas igualmente válidas. Deste modo produz incerteza quantos aos conceitos, regras e princípios que devem ser usados e normalmente são encontrados no cotidiano profissional.

2.2 Abordagens de Ensino

Dentre as diversas abordagens e formatos encontradas na literatura, para prover melhorias no ensino de algoritmos destacam-se as seguintes abordagens:

2.2.1 Abordagem Tradicional

Em uma abordagem de ensino tradicional as aulas, são geralmente realizadas com a apresentação de conteúdos e a exibição de exemplos. Comumente identifica-se a presença de alguma dinâmica de laboratório para que os estudantes possam realizar atividades práticas. Entretanto, essas práticas se tornam complexas de serem gerenciadas e mantidas devido a necessidade do atendimento individual.

A abordagem tradicional de ensino de programação funciona por meio de materiais estáticos como livros, notas, quadro negro e apresentações de slides, etc. que não são suficientes para esse propósito. Eles são muito úteis para apresentar um produto, por exemplo um programa pronto, mas não para apresentar o processo dinâmico de construção de um programa. Assim, mostrando apenas a solução ideal em um processo que não é linear.

O trabalho de Linder et al. (2001) relata que a educação superior tem uma tendência a priorizar os modelos de interação de ensino tradicionais, quando os estudantes estão lendo, ouvindo e assistindo as aulas. Contudo, destaca que para melhorar esse processos de interação devemos priorizar o "Fazer e Discutir", pois promove melhores taxas de

aproveitamento e colaboração com outros estudantes.

2.2.2 Visualização de Programas

A Programação Visual possui o mesmo objetivo de uma linguagem de programação tradicional, diferenciando-se por fazer uso de recursos visuais como Fluxogramas, blocos de código ou representações para projetar o algoritmo. Uma definição é apresentada em Myers (1990) como um sistema que permite ao usuário representar um programa em duas (ou mais) dimensões. Complementando que linguagens de programação tradicionais, baseadas em texto, não são consideradas de duas dimensões, pois compiladores e interpretadores fazem o processamento como uma cadeia de caracteres unidimensional.

De acordo com Yehezkel (2002), a visualização pode favorecer a aprendizagem, desde que a ferramenta utilizada crie uma interação adequada entre o aluno e a visualização. As principais formas de visualização utilizadas no ensino de programação podem ser destacadas de acordo com Price, Baecker e Small (1998):

- Visualização de Programas - esta área tem como objetivo auxiliar o estudante a entender o código, pois é focada na representação gráfica da execução e dos dados;
- Animação do Algoritmo - essa abordagem é focada em mostrar diversas operações realizadas pelo algoritmo, sendo que o usuário pode escolher o que visualizar e em qual o nível de abstração;
- Programação Visual - pode ser caracterizada pelo uso de componentes visuais para construir o programa, sem necessariamente fazer uso de representações textuais.

2.2.3 Robôs

O conceito que aparece na literatura como robótica educativa, consiste basicamente na aprendizagem por meio da montagem de sistemas constituídos por modelos que que podem ser entendidos como robôs. Esses robôs são mecanismos que apresentam alguma atividade física, como movimento de um braço ou movimentar diversos objetos. Ao trabalhar nesse ambiente o objeto de programação passa a ser um dispositivo robótico construído pelos estudantes. Assim, Papert (1983) apresenta como um artefato cognitivo para que os alunos utilizam para explorar e expressar suas próprias ideias, ou "um objeto-para-pensar-com".

O trabalho de Linder et al. (2001) afirmam que a aprendizagem promovida por meio da interação com um ambiente é mais efetiva, ao invés de apenas participar de aulas expositivas. Na década de 1960, foi desenvolvida a linguagem de programação LOGO, projetada inicialmente para ser trabalhada com crianças, este foi o experimento mais popular envolvendo o uso de robôs mecânicos (RESNICK et al., 1996).

Wolz (2001) argumenta que o feedback imediato ofertado pelos ambientes de desenvolvimento integrado, deixa os estudantes muito dependentes da tecnologia. Assim, os

estudantes passam menos tempo planejando seu trabalho, pois partem para a abordagem de tentativa e erro. Já no uso dos robôs os estudantes passavam a ver o planejamento como indispensável, o que os leva, obrigatoriamente, a pensar primeiro em contra ponto a abordagem de tentativa e erro.

2.2.4 Aprendizagem Baseada em Problemas (PBL)

No trabalho Boud e Feletti (1997) encontramos a definição de este tipo de aprendizagem não sendo apenas a inserção de atividades de resolução de problemas em um currículo tradicional, mas como sendo o planejamento da disciplina por meio de problemas-chave que proporcionem uma melhor experiência prática e profissional.

Os principais aspectos do ensino e aprendizado que a aprendizagem baseada em problema tenta alcançar segundo Boud e Feletti (1997) são: resolução de problemas, auto avaliação, retenção de conteúdo, trabalho em grupo e colaboração, criatividade e aprendizado autodidata. Para tanto, podemos destacar essas características descritas no trabalho (NEWMAN, 2005):

Orientação o professor assume um papel de facilitador, mediando o processo de aprendizado, de modo a promover intervenções de acordo com a maturidade e dificuldade dos problemas;

Organização a abordagem trabalha com os problemas sendo elementos inicias no processo de aprendizado, ou seja, o foco não é só trabalhar a resolução de problemas, e sim trabalhar em conjunto com os elementos do processo tradicional;

Cenário o problema objetiva contextualizar o conhecimento estudado, assim agregando o escopo e os conhecimentos que julgar necessário para resolve-lo, deste modo aplicando a teoria;

Grupo o trabalho em grupos pequenos facilita o gerenciamento de vantagens como o compartilhamento de conhecimento e desvantagens como conflitos de ideias e competição entre os participantes.

Para Sakai e Lima (1996) PBL é o eixo principal do aprendizado teórico do currículo de algumas escolas, que a filosofia pedagógica é o aprendizado centrado no aluno, baseando-se no estudo de problemas propostos com a finalidade de fazer com que o aluno estude determinados conteúdos. Embora não constitua a única prática pedagógica, predomina para o aprendizado de conteúdos cognitivos e integração de disciplinas. Esta metodologia é formativa à medida que estimula uma atitude ativa do aluno em busca do conhecimento e não meramente informativa como é o caso da prática pedagógica tradicional.

2.2.5 Teste de software

O teste de software é uma das atividades de verificação e validação mais utilizadas (PRESSMAN, 2011). Segundo Myers, Thomas e Sandler (2004), o objetivo da atividade de teste é revelar a presença de erros ou defeitos no produto. Uma boa atividade de teste é aquela que consegue determinar casos de teste que façam com que o programa falhe.

Em geral, a atividade de teste é realizada em três fases (PRESSMAN, 2011): (1) teste de unidade, que consiste no teste individual de cada módulo de programa, em que deverão ser identificados erros de lógica e programação; (2) teste de integração, o qual se concentra na comunicação entre os módulos do sistema, visando a identificação e correção de erros na interface dos módulos; e (3) teste de sistema, o qual consiste em verificar se todos os módulos combinam-se adequadamente e se a função/desempenho global do sistema é atingida na perspectiva do usuário.

O Ensino Dirigido a Testes (TDL - Test-Driven Learning) é uma proposta de ensino de programação onde os novos conceitos são introduzidos e explorados por meio de testes de unidade automatizados. Foi proposto visando motivar o uso de teste automatizado tanto na atividade de projeto como na atividade de verificação (DESAI; JANZEN; SAVAGE, 2008). Uma das vantagens do TDL é que ele pode ser aplicado em diferentes níveis de um currículo de ciências de computação, desde alunos iniciantes até estudantes mais experientes.

2.3 Trabalhos Correlatos

Dentre os trabalhos, foram selecionados os que mais tinham maior grau de proximidade com o nosso, de forma a prover inspirações e bons direcionamentos.

2.3.1 SOLVEIT

O estudo de Deek (1997) propôs um modelo para resolução de problemas e desenvolvimento de programas denominado Common Model for Problem Solving and Program Development. Este modelo é constituído por seis fases que lembram o ciclo tradicional de desenvolvimento de software – formulação do problema, planejamento da solução, projeto da solução, tradução da solução para uma sintaxe específica, testes e entrega da solução.

Na fase de formulação do problema consiste em identificar e extrair do enunciado as informações relevantes para a resolução do problema. Ele sugere que os estudantes reflitam e questionem sobre as seguintes partes: objetivos, dados, incógnitas, condições e restrições do problema. Estes elementos compõem os dados relevantes do problema e baseiam-se fortemente no trabalho de Polya (1978). Os testes são atividades localizadas na penúltima fase do modelo de resolução de problemas e são utilizados para avaliar se o programa está correto.

Para dar suporte as atividades estabelecidas no modelo, ele construiu um ambiente denominado SOLVEIT (Specification Oriented Language in Visual Environment for Instruction Translation). De modo resumido, esse ambiente oferece editores para os estudantes documentarem suas atividades à medida que seguem nas fases do modelo. A Figura 2.1 mostra os recursos do SOLVEIT para a fase de formulação do problema. Para cada problema, os estudantes devem preencher as informações requeridas – goal, givens, unknowns, e demais sucessivamente. O ambiente oferece um editor que com templates para uma linguagem algorítmica especifica do SOLVEIT, consistindo de tipos básicos de dados, estruturas de controle, entre outros. Entretanto, o ambiente não conta com recursos para compilação e execução de testes.

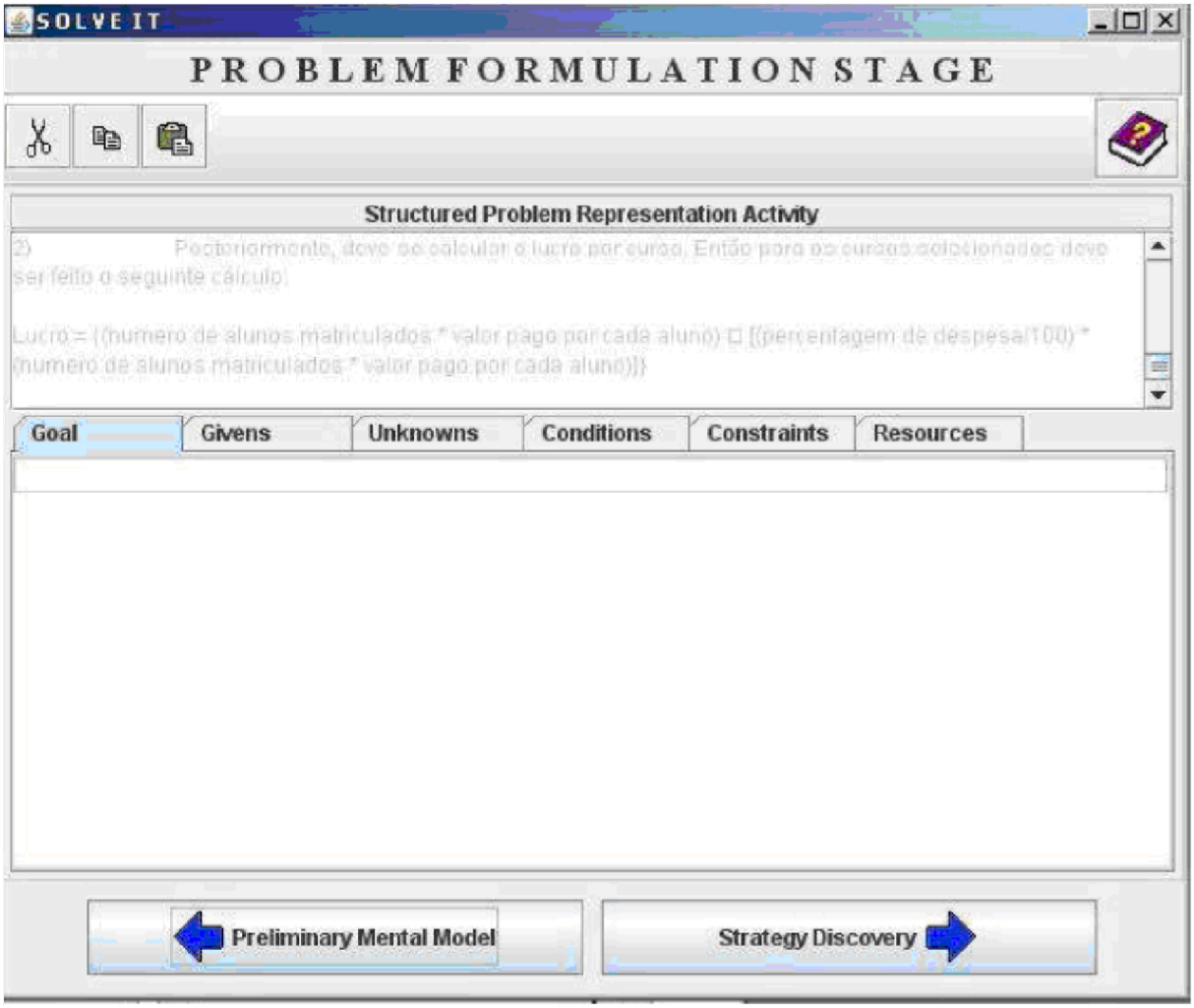


Figura 2.1: Editor do SOLVEIT para a fase de Formulação do Problema. (DEEK, 1997)

Considerando a avaliação, o autor realizou experimentos para avaliar a efetividade do seu ambiente. Assim, a fase de formulação do problema definiu uma escala de pontos para avaliar o entendimento do estudante. Por exemplo, 4 pontos no caso de excelente representação do problema, isto é, o estudante descreveu todos os objetivos, dados, in-

cógnitas e condicionantes; 3 pontos para o caso do estudante representar quase todas as informações relevantes e assim sucessivamente.

2.3.2 Fact-Based

Em Eastman (2003) foi estabelecido uma técnica denominada Fact-Based Problem Identification para orientar os estudantes a identificarem as informações relevantes de um problema. A técnica deve ser aplicada nos casos em que o enunciado do problema é exposto em uma forma narrativa que contém além das informações relevantes, outras informações que estão presentes no enunciado apenas para deixá-lo mais interessante. Ele relata que essas informações extras, geralmente, confundem os estudantes e dificultam o entendimento do problema.

A técnica de Eastman (2003) consiste basicamente em identificar e registrar todos os fatos contidos no enunciado que são relevantes para a resolução do problema. Assim, o termo "fato" faz referência a palavras, nomes, números ou outras expressões que definem alguma propriedade considerada na análise do problema. E define uma entidade com sendo algo que existe física ou abstratamente. De modo que os fatos são identificados, eles devem ser registrados e organizados em uma lista de fatos e com base nela os estudantes fazem a análise do que é solicitado no problema.

Esta estratégia de ler o enunciado, isolar as partes principais do problema e fazer um registro de modo adequado é fundamentada no trabalho de Polya (1978). Ela parte da ideia de que todas as informações estão contidas no enunciado do problema, bastando apenas identificar e isolar o que é relevante para construção da resolução.

2.3.3 JavaTool

O JavaTool apresentado em Mota, Pereira e Favero (2008), é uma ferramenta para auxílio ao ensino de programação, para ser utilizada nas disciplinas iniciais de programação, nas aulas práticas de laboratório. A ferramenta JavaTool foi integrada a um ambiente de ensino Web o Moodle, podendo assim ser utilizada como uma ferramenta de um laboratório virtual de programação. No ambiente Web, o estudante poderá (1) examinar a animação de um exemplo de algoritmo presente no conteúdo ministrado pelo professor e /ou poderá (2) selecionar um exercício proposto pelo professor e desenvolver uma solução. O estudante poderá editar código Java, compilar, depurar o código e visualizar a animação do código.

Uma das ideias centrais do ambiente é animar códigos de algoritmos, escritos com uma sintaxe reduzida da linguagem Java em um curso inicial de programação, sem abordar conhecimentos relacionados à programação orientada a objetos. A ferramenta tem como principal objetivo apoiar estudantes de forma didática durante o início da aprendizagem de programação utilizando a linguagem Java. Isto é proposto através do método de

visualização de programas, em forma de animações 2D, fazendo com que a abstração dos códigos de algoritmos seja visível para o aprendiz.

Na Figura 2.2 é apresentada a interface do JavaTool, com um menu de funções básicas como: criar, abrir, salvar, editar arquivo, imprimir, um menu de ajuda, entre outras. A inserção do código é realizada na área de EDIÇÃO e os resultados de saída do programa, além de possíveis erros que poderão ser exibidos na saída do CONSOLE. A partir do momento que o usuário aciona o botão do play, o código passa a ser animado. Ele conta com uma área de HISTÓRICO corresponde a uma descrição textual do que ocorreu durante os passos da animação. A área ANIMAÇÃO também exibe controles que permitem transitar no histórico por meio dos botões no menu de animação com seta para direita e seta para esquerda.

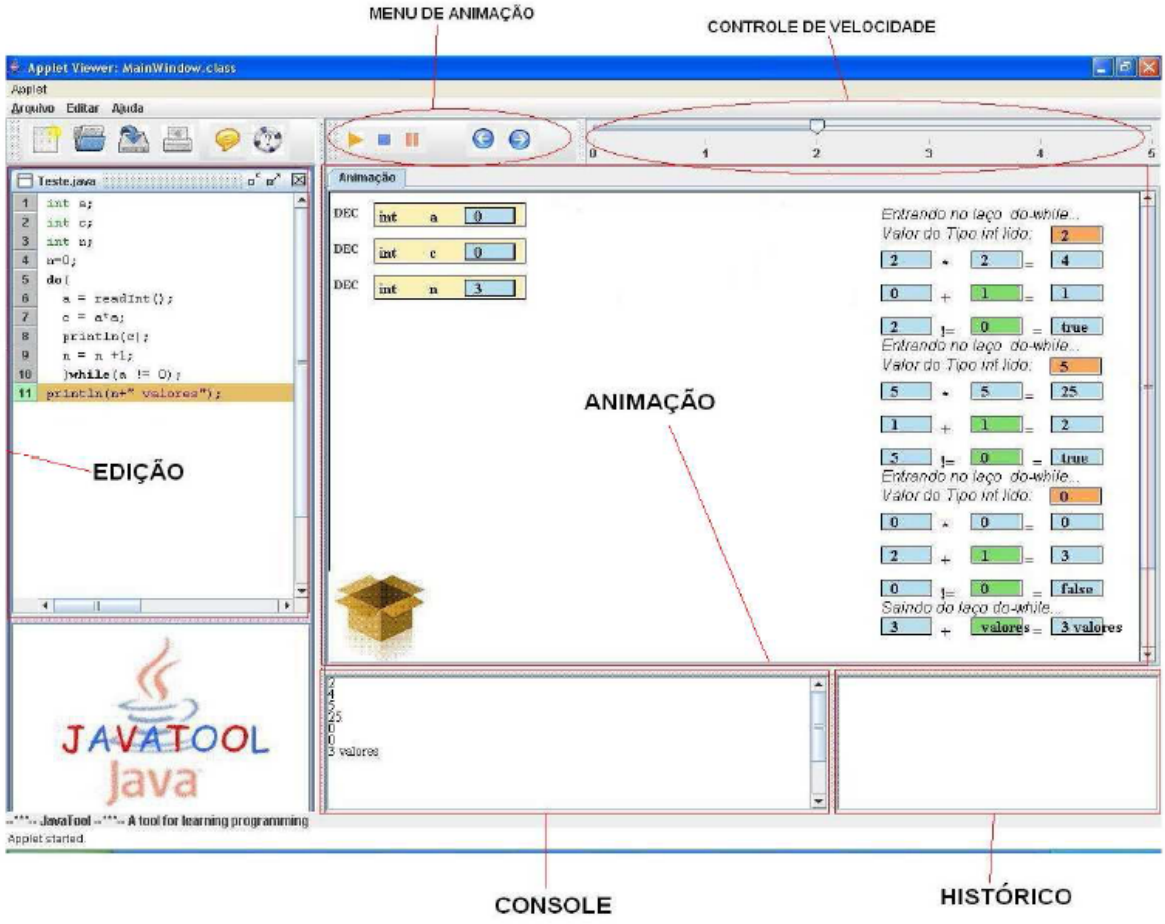


Figura 2.2: Interface do JavaTool. (MOTA; PEREIRA; FAVERO, 2008)

O JavaTool foi desenvolvido para minimizar as dificuldades dos alunos em disciplinas de algoritmos e programação, por meio da animação da execução do código. A ferramenta busca tornar possível a visualização do comportamento da execução do código fonte (MOTA; PEREIRA; FAVERO, 2008). Desta forma, contribui para o desenvolvimento do aluno nas fases iniciais do aprendizado de algoritmos com feedback sobre o que

está desenvolvendo e o que ocorre no código ilustrando passo a passo as execuções.

Contando como principais funcionalidades: edição de código Java, compilação, depuração, visualização da animação do código, exibição dos dados em vários formatos (binário, hexadecimal e octal) e a visualização do histórico de execução da animação tanto gráfico como textual.

2.3.4 Petcha

O trabalho de Queirós e Leal (2012) apresenta uma ferramenta chamada Petcha (acrônimo para Programming Exercises TeaCHing Assistant), que atua como um assistente de ensino automatizado em cursos de programação. O objetivo final do sistema é aumentar o número de exercícios de programação resolvidos pelos alunos.

De acordo com os autores, ele atende a esse objetivo ajuda tanto os professores na criação dos exercícios de programação quanto os estudantes na elaboração das soluções. Além de coordenar uma rede de sistemas heterogêneos, integrando avaliadores automáticos de programas, sistemas de gestão de aprendizado (LMS – Learning Management System), repositórios de objetos de aprendizagem e ambientes de programação integrados.

O foco da ferramenta é servir como complemento para outras aplicações, de modo que ela possa ser removida facilmente. Nesse sentido, ela não fornece um ambiente de desenvolvimento de código, mas sim uma integração com IDEs (Integrated Development Environments) como Eclipse e Visual Studio.

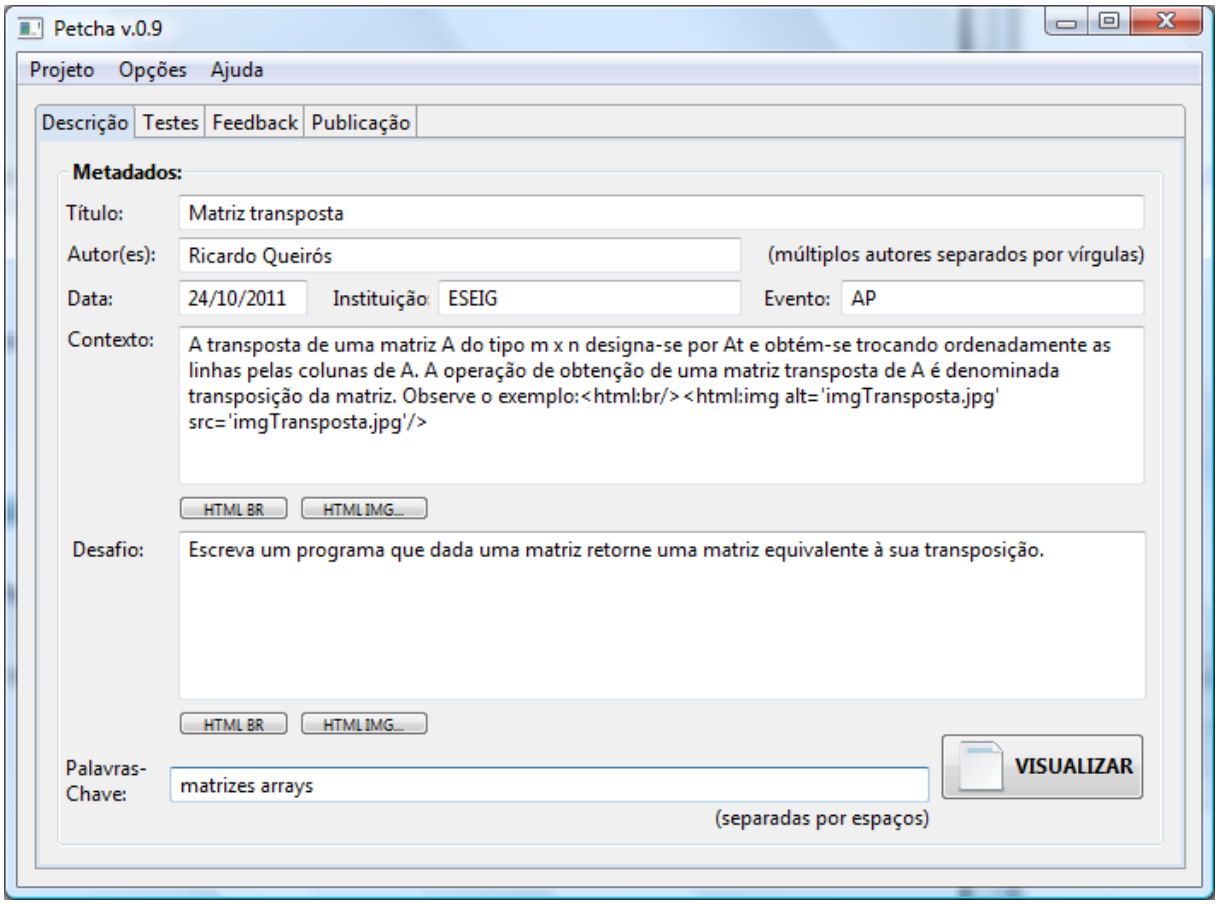


Figura 2.3: Tela de definição do problema no Petcha. (QUEIRÓS; LEAL, 2012)

O estudante deve escolher um exercício, utilizar sua IDE para desenvolver o código e testá-lo, podendo inclusive utilizar dados de teste disponibilizados pelo professor. Ao terminar, o aluno pode submeter seu programa e verificar o feedback textual fornecido pelo programa, podendo inclusive fazer submissões múltiplas.

Os autores conseguem resultados interessantes com um experimento prático, em que a diferença entre os grupos não é grande, mas estatisticamente significante.

2.3.5 iProgram

Em Neto (2015) a ferramenta disponibiliza ao professor o gerenciamento de turmas, alunos, exercícios e questões, de maneira que cabe a ele organizar o seu trabalho com o apoio do iProgram. Para elaborar exercícios, o professor pode utilizar um banco de questões já cadastradas no sistema, o que pode agilizar seu trabalho e permitir o reaproveitamento de questões em turmas diferentes, por exemplo.

Cada exercício pode ser composto por questões de seis tipos. Essa variedade de questões não só permite que o professor possa avaliar diferentes tipos de habilidades em seus alunos, como também possibilita que sejam elaborados exercícios com diferentes níveis de dificuldade. A cada questão do exercício é associado um objetivo de aprendizagem. Essa

associação permite que o professor possa acompanhar de perto como anda o desempenho dos seus alunos com relação às habilidades que se espera que eles desenvolvam.

Os exercícios possuem correção automática e manual, proporcionando feedback imediato, importantes no aprendizado de programação. Além disso, a correção semiautomática permite também que o professor possa analisar ele mesmo as respostas dos alunos e possa prover um feedback mais detalhado

A ferramenta permite que haja uma troca de feedback entre professor e aluno em cada questão do exercício. Assim, a intensão era diminuir a distância entre aluno e professor para que pudesse haver uma melhora no aprendizado. Além de a ferramenta oferecer estatísticas para professor e aluno através de relatórios gráficos.

Esses relatórios gráficos, no contexto do professor, permitem que ele possa avaliar suas turmas ou estudante de maneira isolada. Assim, o professor pode visualizar o desempenho da sua turma por exercício, por tipo de questão e por objetivo de aprendizagem. Dessa forma, é possível identificar quais objetivos estão sendo atingidos e quais precisam ser mais bem trabalhados pelo professor.

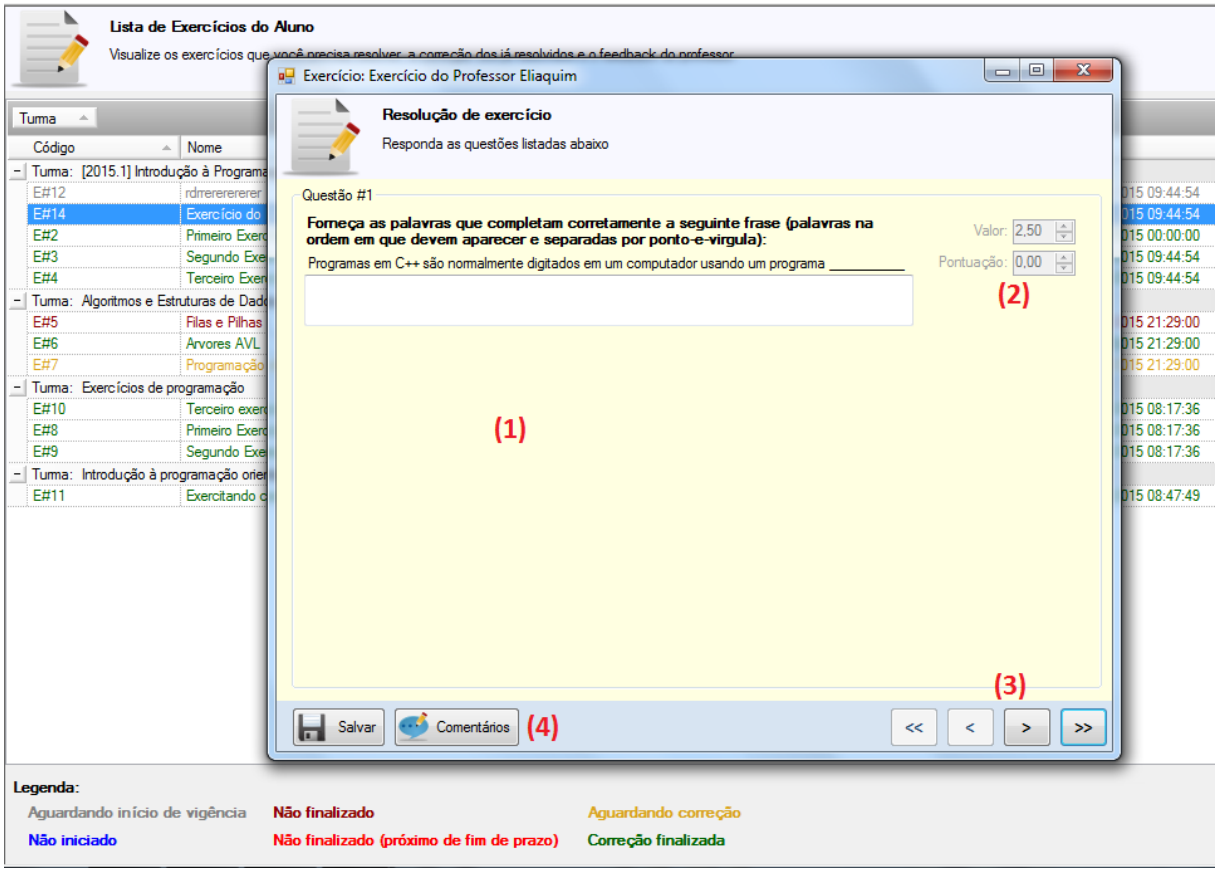


Figura 2.4: Tela de resolução de exercício no iProgram. (NETO, 2015)

A tela de resolução de exercício é dividida da seguinte forma: em (1) são exibidas as questões; em (2) o sistema mostra qual o valor da questão sendo exibida e qual a pontuação para aquela questão (nesse momento, como o exercício ainda não foi corrigido,

a pontuação sempre será zero); em (3) o aluno pode navegar pelas questões do exercício e, a questão que está sendo exibida em (1) muda a visualização; em (4), se a questão permitir feedback por parte do professor, o aluno pode clicar no botão Comentários e escrever alguma mensagem para o professor, que poderá lê-la ao corrigir o exercício.

Além disso, há também o botão Salvar, que quando clicado, o sistema salva a resolução do exercício e pergunta se o Aluno deseja submeter, naquele momento, o exercício para correção. Em caso afirmativo, o exercício passa para o estado Aguardando Correção. As questões corrigidas automaticamente já ficam disponíveis para o aluno visualizá-las, e o exercício poderá ser corrigido pelo professor.

2.3.6 POPT

Em Neto et al. (2013) trabalho foi apresentada uma metodologia de ensino de programação orientada a problema e testes para disciplinas de introdução a programação, chamada POPT (do inglês: Problem Oriented Programming and Testing). Esta metodologia é apoiado por uma ferramenta chamada TestBoot, também desenvolvida no contexto do trabalho.

A ferramenta TestBoot foi implementada em C++ e que implementa os conceitos para capacitar os estudantes iniciantes na definição de casos de testes de maneira simples. Os casos de testes são representados na forma de dados de entrada e saída de casos de testes em uma simples tabela, não exigindo que os alunos aprendam APIs de frameworks de teste ou alterem a estrutura do programa para permitir o uso de assertivas embutidas no código, como proposto por outras abordagens, e que podem dificultar a adoção de testes por alunos iniciantes.

Foi feito um estudo de caso em uma turma iniciante do curso de Ciência da Computação, durante o primeiro Semestre de 2012. Os alunos foram divididos em dois grupos, e apenas um grupo foi exposto aos novos conceitos de POPT e TestBoot. Ambos grupos receberam um problema e desenvolveram uma solução para ele.

Apesar de, no final do estudo, os valores finais de qualidade para os dois grupos terem sido equivalentes, além de implementar o programa, cada aluno implementou também em média cerca de 16 casos de testes. Além disto, eles apresentaram menos versões que erros e passaram mais tempo para apresentar a primeira versão, alcançando nesta primeira versão uma maior qualidade quando comparados aos outros estudantes. Os autores creem que POPT estimula os alunos a pensar melhor sobre o problema proposto e sobre a solução.

Capítulo 3

O Método: Composição de Resultado

O método de Composição de Resultado é um caminho para a resolução de problemas de programação, possibilitando o estudante iniciar(sair da inercia) e progredir na resolução de problemas, de modo que ele e seu professor possam acompanhar a evolução das suas tentativas. Portanto, considera o problema sendo dividido em partes, assim o estudante pode obter um acerto parcial no desenvolvimento da sua resposta. Por outro lado, este método estimula o estudante a "sair da inercia", problema muito comum nos estudantes de programação, dar um primeiro passo.

3.1 Fundamentos

O objetivo da Composição de Resultado é oferecer um método de resolução de problemas de programação. Visto que o processo de resolução de um problema e bastante complexo e abstrato, e muitas vezes é definido como “criativo” (BENNEDSEN; CASPERSEN, 2004). Para tanto, este método oferece um conjunto de passos visando entender, iniciar e progredir até a resolução total de um problema.

A Composição de Resultado foi concebida a partir de ideias como o método de Divisão e Conquista, o Test Driven Development (TDD) e observações de como alunos resolvem problemas em sala.

Divisão e conquista é uma técnica para projeto de algoritmo, nela um problema é dividido em subproblemas menores do problema original. A conquista trata da resolução dos subproblemas se possível, caso contrário podem ser necessárias novas divisões. E por fim é realizada a combinação das soluções dos subproblemas para o problema original (CORMEN et al., 2012).

O Test Driven Development é uma técnica de desenvolvimento de software baseada em um ciclo curto de repetições, em que são definidos testes, são criadas funcionalidades, assim como são realizados de mudanças para aperfeiçoar o código, ambas avaliadas pelos

testes (BECK, 2003).

O método de Composição de Resultado pode ser descrito nos seguintes passos:

1. Dividir o problema
2. Conceber a base do programa
3. Resolução dos subproblemas
   3.1 Construir a resposta para um subproblema
   3.2 Realizar um ou mais testes focados nesse subproblema
4. Integrar as respostas dos subproblemas

3.1.1 Passo 1: Dividir o problema

As abordagens tradicionais de ensino de programação não apresentam um método formal, que ajude o estudante a construir a estrutura necessária para a resolução dos problemas. Assim muitos estudantes nem sabem por onde começar, devido a enorme abstração entre a definição do problema e a implementação da resposta (BENNEDSEN; CASPERSEN, 2004). Este é o primeiro passo, pois é necessário fazer com que o estudante pense no problema. Além disso, o educador deve acompanhar o estudante mostrando exemplos e auxiliando na segmentação do problema, indicando as possibilidades de desdobramentos.

A divisão do problema é orientada pelos possíveis subproblemas que ele engloba. Visto que, geralmente, eles recebem valores como entrada, processam os dados e informam os resultados, conhecida como saída do programa. Essa divisão baseia-se na análise do resultado esperado, e decompondo essa análise inicial em subproblemas, ou seja, cada parte da saída pode se tornar um componente do problema original (subproblema). Assim, servindo como base para a realização continua da resolução e combinação dos subproblemas: a saída vai sendo composta e o problema integralmente resolvido.

Esse passo de divisão visa oferecer treino na habilidade de entendimento de um problema, subdividindo em partes mais simples. Para que o estudante possa identificar quais itens do problema vão lhe levar a solução completa. Deste modo, identificando as partes que sabe resolver e qual é mais "fácil"para iniciar a solução, além de compreender os conhecimentos necessários para desenvolver os demais subproblemas.

Identificação de elementos importantes para resolução

Como mecanismo para facilitar a execução do Passo 1, criamos uma ficha, que tem por objetivo auxiliar o processo de divisão do problema. Inicialmente é feito um trabalho de identificação: variáveis de entrada e de saída. Assim como estruturas pertinentes a construção da resposta, como a identificação de estruturas de controle como as de seleção e de repetição. Essa ficha está Apêndice A: Ficha de Avaliação do Problema.

Usar essa ficha de avaliação é uma tentativa de "burocratizar" o processo de construção da resposta, e assim promover o pensamento da resolução do problema. Deste modo, é uma tentativa de evitar que o estudante inicie a codificação sem ter um entendido básico do enunciado do problema. Sem ter feito um exame detalhado prévio.

O preenchimento inicia pela identificação do problema e o nome que vai ser dado ao programa construído. Seguido pela identificação prévia das variáveis de entrada, entretanto, esse item não possui a pretensão de indicar todas as variáveis utilizadas, mas sim indicar um ponto de partida.

Seguimos indicando os elementos do Item 2 que trata dos valores de saída, orientamos que o estudante faça uso de variáveis para armazenar o resultado. Assim, facilitando as próximas Passos. O Item 3 busca obter informações sobre o cálculo que o programa deve realizar: fórmulas matemáticas, restrições e possíveis desdobramentos do problema.

Os Itens 4 e 5 trabalham a identificação de estruturas de controle, com um enquadramento para cada tipo de estrutura, a de seleção podendo ser identificada com perguntas, restrições e critérios. Já a estrutura de repetição é enquadrada por um instruções que devem ser executadas várias vezes. Em ambas as estruturas existe um pequeno resumo para auxiliar o estudante a escolher qual estrutura que deve usar, considerando as especificidades de cada uma.

Vamos considerar o seguinte exemplo:

Problema do Maior: Construa um programa que informe o maior valor entre três números lidos.

Segue na próxima página a ficha que foi preenchida para exemplificar os elementos de cada item e como deveria ser preenchida pelos estudantes, de modo que eles possam fazer uma análise do problema e identificar os seus elementos:

Ficha de Avaliação do Problema

Problema: MAIOR DE TRÊS

Nome do Programa: MAIORTRES

Item 1: Destaque as variáveis de **Entrada**:

Descrição	Tipo (int, float, char, String, boolean, etc...)	Nome da Variável	Valor Inicial
Primeiro Num.	int	a	0
Segundo Num.	int	b	0
Terceiro Num.	int	c	0

Item 2: Destaque os Valores de **Saída**:

Descrição	Ex: Tipo ou Formatação de saída
Maior das três => x	int ; x = 0

Item 3: Identifique e o **cálculo** que o programa deve realizar (Baseado nos elementos de entrada/saída):

Comparações lógicas: > maior, < menor

a => maior ; b => meio ; c => maior

Item 4: O programa apresenta alguma **pergunta** que deve ser respondida (Decisão, Critério, Restrição)?
Indique(circule) qual estrutura de **SELEÇÃO** deve ser usada?

Seleção Simples	Seleção Dupla/Composta	Seleção Múltipla

if (numero > 7) {
  System.*out*.println("maior que 7");
}
	if (numero > 7) {
	  System.*out*.println("maior que 7");
	} else {
	  System.*out*.println("<= a 7");
	}
		switch (opcao) {
		  case 1:
		  System.*out*.println("Opcao 1");
		  break;
		  default:
		  System.*out*.println("Opcao invalida");
		}

Uma pergunta com uma caminho possível de execução.	Uma pergunta com dois caminhos de execução. **Nota**: Se houver mais de uma pergunta dependente, será necessário encadear if e else.	Seleciona um grupo de ações dependendo da avalição de valor de opção, oferecendo múltiplos caminhos de execução. **Nota**: Não consegue trabalhar com faixas de valores.

Item 5: O programa apresenta alguma **instrução** que deva ser **executada mais de uma vez**?
Indique(circule) qual estrutura de **REPETIÇÃO** deve ser usada?

Repetição: while	Repetição: do..while	Repetição: for

// Controlada por Contador
int i = 1; // Criacao var. controle
while (i <= 10) { // teste de continuidade
  System.*out*.println(i);
  i++; // incremento do controle
}
// Controlada por Flag
while (numero != -1) {
  System.*out*.println(i);
  numero = entrada.nextInt();
}
	int i = 1;
	do {
	  System.*out*.println(i);
	  i++;
	} while (i <= 10);
		for (int i = 0; i <= 10; i++) {
		  System.*out*.println(i);
		}

Resolve qualquer problema de repetição, entretanto é mais indicado para repetição controlada por Flag	Semelhante ao laço while, mas executa as instruções pelo menos uma vez e possui o teste lógico no final.	Laço especialista em repetição controlada por Contador

3.1.2 Passo 2: Conceber a base do programa

A concepção da base do programa tem como objetivo estabelecer o mínimo de código necessário para dar início à resolução do problema. Portanto, este deve possuir as variáveis mais básicas encontradas na descrição do problema, assim apresentando como resultado valores triviais de resposta para o problema analisado, além de comandos básicos de entrada e saída de console.

Nessa etapa, o programa recebe uma entrada vazia, por exemplo composta por valor (es) zero(ou nulos), devendo produzir uma resposta nula para a entrada vazia; ou seja, deve-se indicar a entrada e saída dos valores mais triviais possíveis, de acordo com o domínio do problema. Podendo ter um ou mais valores nulos, respeitando as definições do problema.

Nesse contexto, é esperado que o estudante possa produzir os códigos iniciais para um teste trivial. Além disso, é esperado que ele possa testar entradas com valores de saídas conhecidas, pois essa análise é importante para resolver partes do problema de forma isolada, pensando em entradas que favoreçam o teste de um determinado subproblema.

Assim, a base do programa deve ter um comando de entrada de dados que lê os três números, e um comando de saída para indicar a resposta do programa. Sempre é interessante usar variáveis para guardar o resultado. Deste modo, essa base simples da resolução já está pronta para passar em um teste inicial, sendo o mais trivial possível, ou seja, com entradas e saídas nulas:

a)

Entrada: 0 0 0

Saída: 0

3.1.3 Passo 3: Resolução de um subproblema

Esse passo é orientado por testes, por exemplo, descobrindo a resposta para casos de testes simples que favorecem a checagem de um subproblemas. A partir do Passo 2: Conceber a base do programa, pode-se testar a entrada nula e obter a saída nula; garantindo o início correto da resolução.

Considerando o **Problema do Maior:** temos uma entrada de valores para A, B e C e devemos produzir como resposta (saída) X. Então neste ponto podemos ter uma entrada com valores “0 0 0” correspondente à entrada vazia e produziremos um valor nulo como X=0, ou seja, a resposta vazia para entrada vazia. Agora, podemos pensar a resolução de um subproblema que possua menor complexidade.

Aqui podemos destacar três subproblemas considerando o problema exemplo, orientados pelas possibilidades de formar a saída X, ou seja, A sendo maior e que B e C, B sendo maior que A e B, e C sendo maior que A e B.

Após encontrar uma implementação para o subproblema A, um bom caso de teste deve ser usado para averiguar se esse subproblema foi resolvido. O candidato ideal para o teste seria um valor de A que consiga isolar X, ou seja, produzir a resposta que considere somente entrada A. Isto se faz necessário para segmentar a resolução do problema e deixar o restante, B e C para os próximos passos.

Logo após a resolução do subproblema que produz a resposta que depende de A, deve-se prosseguir para o próximo item de menor complexidade. Considerando como próximo subproblema o item B, os testes devem ser direcionados para entradas que favoreçam o teste que produz exclusivamente B, e produzam valores nulos para os outros itens A e C.

Em caso de falha na resolução o estudante possui duas alternativas: primeiro deve buscar o conhecimento necessário e depois pode descartar a porção de código que apresenta dificuldade.

Nesse ponto, o estudante possui um roteiro para cada problema, do que deve realizar e os testes indicam se está conseguindo atingi-los. Caso não consiga produzir a resposta correta, pode oferecer uma resposta parcialmente correta.

O foco desse passo é oferecer uma alternativa ao paradigma tradicional de resolução de problemas, que exercita e mostra várias respostas das atividades, esperando que o estudante possa usar em outras soluções. Assim, o estudante pode pensar a resolução de um problema de maneira crítica, identificando dificuldades e apresentando maneiras de se resolver subproblemas, que possam fornecer um viés para que o estudante possa conseguir sua autonomia.

3.1.4 Passo 4: Integrar as respostas dos subproblemas

A integração dos subproblemas já ocorre nas sucessivas execuções do Passo 3: Resolução de um subproblema, de modo implícito, entretanto não a percebemos por conta dos testes que são focados em um determinado subproblema.

Para tanto, é necessário realizar novos testes com entradas que favoreçam a resposta conjunta dos subproblemas, averiguando se as partes continuam produzindo a resposta correta.

E deste modo realizar alterações necessárias para o funcionamento das partes em conjunto. Até mesmo identificando oportunidade de melhorar o código da resposta.

3.2 Tipos de Resultados

Como complemento dos passos do método, é necessário refletir acerca dos possíveis desdobramentos de um problema, que darão origem aos subproblemas. Essa análise é centrada no resultado esperado como resposta para um dado problema.

Para tanto, é indispensável que no enunciado do problema sejam definidos como devem ser apresentadas as entradas e saídas, preferencialmente oferecendo exemplos de pares

de entrada/saída. Deste modo, minimizando dúvidas de como o problema é tratado e oferecendo exemplos de entradas/saídas para o estudante refletir sobre o algoritmo.

Para o melhor entendimento da divisão em subproblemas temos que realizar as definições sobre a saída, que apresentam em sua resposta elementos que podemos classificar como estendida ou reduzida. Pode haver uma abordagem híbrida que relacione elementos característicos de ambas, podendo aumentar a complexidade da tarefa de compreensão do problema e implementação da resposta.

3.2.1 Saída Estendida

Os problemas que apresentam a Saída Estendida possuem uma ou mais variáveis de entrada, desta forma, o elemento de entrada gera vários elementos da saída. Considerando um exemplo hipotético de lermos uma varável Z e temos que uma saída com A, B e C é classificado como de Saída Estendida, assim devendo produzir uma saída formada pelos elementos "A B C".

A Saída Estendida pode oferecer mais vantagens didáticas, pois ela vai trabalhar a mesma variável de entrada para produzir diferentes partes do resultado. Assim poderíamos ter como desdobramentos os subproblemas:

Primeiro - Produzir a resposta para entrada vazia, ou seja Z = 0 e a saída produzida seria "0 0 0 ";

Segundo - Buscar uma implementação para o item A da resposta e testar com a entrada que produza, preferencialmente, "A 0 0"como resposta;

Terceiro - Buscar uma implementação para o item B da resposta e testar com a entrada que produza, preferencialmente, "0 B 0"como resposta;

Quarto - Buscar uma implementação para o item C da resposta e testar com a entrada que produza, preferencialmente, "0 0 C"como resposta.

Se não for possível realizar o teste de maneira isolada, devemos direcionar a análise com foco no subproblema que está sendo resolvido, ou seja abstraindo as demais analises.

Exemplo de Saída Estendida

Segue um exemplo de aplicação do método com saída Estendida:

Problema 01: Faça um programa para controlar a distribuição de notas em um caixa eletrônico, que recebe um valor de entrada para o saque, então sua tarefa é calcular a quantidade de cédulas que deverão ser disponibilizadas, os valores das cédulas são: 50, 10, 5 e 1 em reais. Ofereça cédulas de maior valor e assume que sempre haverá notas disponíveis. A saída deve ser informada em uma única linha com os valores das notas de 50, 10, 5 e 1 respectivamente, ex: "0 0 0 0 ".
a)

Entrada: 0

Saída: 0 0 0 0

b)

Entrada: 100

Saída: 2 0 0 0

Resolução

Definições preliminares:

Saque - valor informado pelo usuário;

Notas50 - número de notas com valor de 50 reais;

Notas10 - número de notas com valor de 10 reais;

Notas5 - número de notas com valor de 5 reais;

Notas1 - número de notas com valor de 1 real.

Passo 1: Dividir o problema Subproblemas:

Primeiro - Produzir a resposta para entrada vazia, ou seja Saque = 0 e a saída produzida seria "0 0 0 0".

Segundo - Buscar uma implementação para o item Notas50 da resposta e testar com entrada que produza, preferencialmente só o valor das notas de 50, como 200 reais que leva a resposta: "4 0 0 0";

Terceiro - Buscar uma implementação para o item Notas10 da resposta, e testar com entrada que produza somente o valor das notas de 10, como 30 reais que leva a resposta: "0 3 0 0";

Quarto - Buscar uma implementação para o item Notas5 da resposta e testar com entrada que produza somente notas com valor 5, como 5 que produz a saída "0 0 1 0"como resposta;

Quinto - Buscar uma implementação para o item Notas1 da resposta e testar com entrada que produza, preferencialmente só o valor das notas de 1, como 4 reais que leva a resposta: "0 0 0 4".

Passo 2: Conceber a base do programa a estrutura inicial proposta para o problema do caixa eletrônico pode ser visualizada na Figura 3.1 , escrito na linguagem de programação java, que indica o mínimo possível de código para iniciar o trabalho de resolução do problema, além de comandos simples de entrada e saída. Assim, exemplificando o código inicial que o estudante deve produzir.

```
1
2  import java.util.Scanner;
3
4  public class CaixaEletronico {
5
6      public static void main(String[] args) {
7          int saque = 0;
8          int notas50 = 0, notas10 = 0, notas5 = 0, notas1 = 0;
9
10         Scanner in = new Scanner(System.in);
11         saque = in.nextInt();
12
13         System.out.println(notas50 + " " + notas10 + " " + notas5 + " " + notas1);
14     }
15 }
16
17
```

Figura 3.1: Estrutura inicial produzida no Passo 2.

O **Passo 3: Resolução de um subproblema** já possui no passo anterior a base para o teste da entrada vazia. Assim, bastando informar a entrada "0"na execução do programa da Figura 3.1 e obtém a saída "0 0 0 0", pois ainda não foi implementado nenhuma etapa de processamento. Nesta perspectiva, já temos a resposta do Primeiro Subproblema, bastando apenas aplicar os comandos mais básicos da linguagem de programação.

A Figura 3.2 representa a resolução do segundo subproblema e seguindo-se as implementações sucessivas até o resultado final apresentado na Figura 3.3.

```java
import java.util.Scanner;

public class CaixaEletronico {

    public static void main(String[] args) {
        int saque = 0;
        int notas50 = 0, notas10 = 0, notas5 = 0, notas1 = 0;

        Scanner in = new Scanner(System.in);
        saque = in.nextInt();

        //segundo subproblema
        notas50 = saque/50;

        System.out.println(notas50 + " " + notas10 + " " + notas5 + " " + notas1);
    }
}
```

Figura 3.2: Implementação da resposta do segundo subproblema.

```java
import java.util.Scanner;

public class CaixaEletronico {

    public static void main(String[] args) {
        int saque = 0;
        int notas50 = 0, notas10 = 0, notas5 = 0, notas1 = 0;

        Scanner in = new Scanner(System.in);
        saque = in.nextInt();

        //segundo subproblema
        notas50 = saque / 50;

        //terceiro subproblema
        notas10 = saque % 50 / 10;

        //quarto subproblema
        notas5 = saque % 50 % 10 / 5;

        //quinto subproblema
        notas1 = saque % 50 % 10 % 5;

        System.out.println(notas50 + " " + notas10 + " " + notas5 + " " + notas1);
    }
}
```

Figura 3.3: Implementação de todos os subproblemas.

Como justificado anteriormente, em cada novo subproblema a ser resolvido deve-se realizar casos de testes, que favoreçam a análise do subproblema. Contudo, nada impede que os testes produzam respostas para todos os subproblemas já implementados. Por

exemplo, avaliando a Figura 3.3 e imaginando que somente os subproblemas das notas de 50 e 10 estivesse implementado, um bom teste seria a entrada para saque seria o valor 90 que produziria a saída "1 4 0 0 ".

Assim o **Passo 4: Integrar as respostas dos subproblemas** vai sendo realizado no decorrer do **Passo 3: Resolução de um subproblema**, restando apenas implementar algum ajuste e realizar testes que envolvam todos os subproblemas.

3.2.2 Saída Reduzida

Os problemas que apresentam a Saída Reduzida possuem mais de uma variável de entrada, realizando algum trabalho que se consolida em uma saída menor, ou seja, recebendo diversas variáveis e produzindo um resultado resumido.

Exemplo de Saída Reduzida

Segue um exemplo de aplicação do método com saída Reduzida:

Problema 02: Construa um programa para calcular o conceito final de um aluno obtendo as notas de três provas realizadas por ele durante o período letivo. Os conceitos a serem aplicado são: EXCELENTE com média igual ou acima de 8, BOM com média de 5 até 7.99 e INSUFICIENTE caso seja menor que 5. A saída deve ser informada em uma única linha com o conceito referente a média das notas obtidas nas provas.
a)

Entrada: 0 0 0

Saída: INSUFICIENTE

b)

Entrada: 10 7.5 5

Saída: BOM

Resolução

Definições preliminares:

Nota_1 - valor da nota da primeira prova;

Nota_2 - valor da nota da segunda prova;

Nota_3 - valor da nota da terceira prova;

Conceito - Conceito a ser informado para o estudante.

Passo 1: Dividir o problema Subproblemas:

Primeiro - Produzir a resposta para entrada vazia, ou seja Nota_1=Nota_2=Nota_3 = 0 e a saída produzida seria "INSUFICIENTE".

Segundo - Buscar uma implementação para o item INSUFICIENTE da resposta e testar com entrada que produzam média menor que 5 e informar o conceito INSUFICIENTE;

Terceiro - Buscar uma implementação para o item BOM da resposta, e testar com entrada que produzam média entre 5 e 7.99 e informar o conceito BOM;

Quarto - Buscar uma implementação para o item EXCELENTE da resposta e testar com entrada que produzam média maior ou igual a 9 e informar o conceito EXCELENTE.

Passo 2: Conceber a base do programa a estrutura inicial proposta para o Problema 2 Conceito Final pode ser visualizada na Figura 3.4, escrito na linguagem de programação java, indica o mínimo possível de código para iniciar o trabalho de resolução do problema, além de comandos simples de entrada e saída.

```java
import java.util.Scanner;

public class ConceitoFinal {

    public static void main(String[] args) {
        String conceito = "INSUFICIENTE";
        double nota1 = 0, nota2 = 0, nota3 = 0;

        Scanner in = new Scanner(System.in);
        nota1 = in.nextDouble();
        nota2 = in.nextDouble();
        nota3 = in.nextDouble();

        System.out.println(conceito);
    }
}
```

Figura 3.4: Concepção da estrutura inicial do programa.

Com a concepção inicial apresentada na Figura 3.4, já é possível executarmos o teste sobre a entrada vazia e produzirmos o resultado do primeiro subproblema.

```java
import java.util.Scanner;

public class ConceitoFinal {

    public static void main(String[] args) {
        String conceito = "INSUFICIENTE";
        double nota1 = 0, nota2 = 0, nota3 = 0, media = 0;

        Scanner in = new Scanner(System.in);
        nota1 = in.nextDouble();
        nota2 = in.nextDouble();
        nota3 = in.nextDouble();

        media = (nota1 + nota2 + nota3) / 3;

        //segundo subproblema
        if (media < 5) {
            conceito = "INSUFICIENTE";
        }

        System.out.println(conceito);
    }
}
```

Figura 3.5: Implementação do segundo subproblema.

Na Figura 3.5 temos a implementação do segundo subproblema, como diferenciais do código anterior, uma nova variável e o seu cálculo simples de média. Desta forma fazendo os ajustes necessários à resolução do subproblema em questão.

```java
import java.util.Scanner;
public class ConceitoFinal {
    public static void main(String[] args) {
        String conceito = "INSUFICIENTE";
        double nota1 = 0, nota2 = 0, nota3 = 0, media = 0;

        Scanner in = new Scanner(System.in);
        nota1 = in.nextDouble();
        nota2 = in.nextDouble();
        nota3 = in.nextDouble();

        media = (nota1 + nota2 + nota3) / 3;

        //segundo subproblema
        if (media < 5) {
            conceito = "INSUFICIENTE";
        }
        //terceiro subproblema
        if (media >= 5 && media <= 7.99) {
            conceito = "BOM";
        }
        //quarto subproblema
        if (media >= 8) {
            conceito = "EXCELENTE";
        }
        System.out.println(conceito);
    }
}
```

Figura 3.6: Implementação de todos os subproblemas.

As sucessivas resoluções de subproblemas, requisitos do **Passo 3: Resolução de um subproblema** levarão as implementações de todos os subproblemas como apresentado na Figura 3.6. Assim como realizar os testes específicos de cada um deles, averiguando um teste mais genérico.

```java
import java.util.Scanner;
public class ConceitoFinal {

    public static void main(String[] args) {
        String conceito = "INSUFICIENTE";
        double nota1 = 0, nota2 = 0, nota3 = 0, media = 0;

        Scanner in = new Scanner(System.in);
        nota1 = in.nextDouble();
        nota2 = in.nextDouble();
        nota3 = in.nextDouble();

        media = (nota1 + nota2 + nota3) / 3;

        //segundo subproblema
        if (media < 5) {
            conceito = "INSUFICIENTE";
        } else if (media <= 7.99) {//terceiro subproblema
            conceito = "BOM";
        } else if (media >= 8) {//quarto subproblema
            conceito = "EXCELENTE";
        }
        System.out.println(conceito);
    }
}
```

Figura 3.7: Ajustes realizados no passo 4.

A execução do **Passo 4: Integrar as respostas dos subproblemas** pode produzir um resultado semelhante ao observado na Figura 3.7, em que ajustes são realizados para melhorar e integrar o código dos subproblemas. Essa solução proposta pode sofrer inúmeras melhorias(Refatoração), mas baseado na ideia que todos os testes anteriores devem continuar funcionando, o estudante pode tentar provê-las.

3.3 Considerações

A Composição de Resultado tenta segmentar a complexidade de problemas, logo, o roteiro descrito pode ser expandido e adaptado, pois os problemas possuem diversos formatos e raciocínios associados. Problemas com várias entradas e saídas diferentes das apresentadas neste modelo podem se beneficiar deste método, focando nas ideias mais gerais como: os desdobramentos do problema, a codificação da estrutura básica e os testes sucessivos.

O **Passo 3: Resolução de um subproblema**, vem a beneficiar a resolução de problemas de programação, pois ele leva o estudante a mapear a resolução em pequenas estruturas que venham a responder os subproblemas. Identificando a parte de maior dificuldade, oportunizando a busca do necessário para supera-las.

O método oferece um mecanismo sistemático de produção de soluções para problemas de programação, oferecendo uma forma de iniciar e progredir a resolução. Deste modo, realiza o acompanhamento do progresso do estudante, possibilitando que os educares possam prover intervenções.

As tarefas do método da Composição de Resultado podem ser consideradas onerosas e com uma curva de aprendizado extra. Contudo, a abordagem apresenta caminhos e possibilidades para que o estudante possa iniciar as resoluções, progredir e acompanhar o seu desempenho por meio dos testes com foco específico. Em caso de adversidades, pode-se diagnosticar as principais dificuldades e buscar soluções.

Capítulo 4

Ambiente de Aprendizagem

Apesar do descrito no Capítulo 3 - O Método: Composição de Resultado, ser uma tentativa de prover melhorias ao ensino e treino de programação, a execução manual do método revela-se bastante trabalhosa. Para tanto, foram desenvolvidas ferramentas de software para auxiliar a sua utilização por educadores e estudantes.

A solução tecnológica que forma o ambiente de aprendizagem é composta por um IDE para o desenvolvimento dos códigos de programação - NetBeans IDE, um sistema web que fornece gerenciamento e apoio - LabProg Web, e uma ferramenta de integração que permite a troca de informações entre o sistema web e o IDE, denominada - Plug-in LabProg.

4.1 NetBeans IDE

O ambiente de desenvolvimento Salter e Dantas (2014) é uma ferramenta de código aberto que permite desenvolver o trabalho de maneira rápida, fácil e com muitos instrumentos de auxílio, além dele possuir uma vasta documentação, exemplos de códigos para extensão, deste modo, facilitando a implementação novas funcionalidades.

Esta ferramenta de desenvolvimento de software permite que o seu usuário trabalhe em diversas plataformas, e oferece suporte para se trabalhar em diversas linguagens, entretanto escolhemos a linguagem de programação Java, pois construir um mecanismo genérico para diversas linguagens seria bastante complexo e trabalhoso.

Contudo, os esforços empregados nesse trabalho poderão ser facilmente reutilizados para estender a ferramenta e oferecer suporte a outras linguagens.

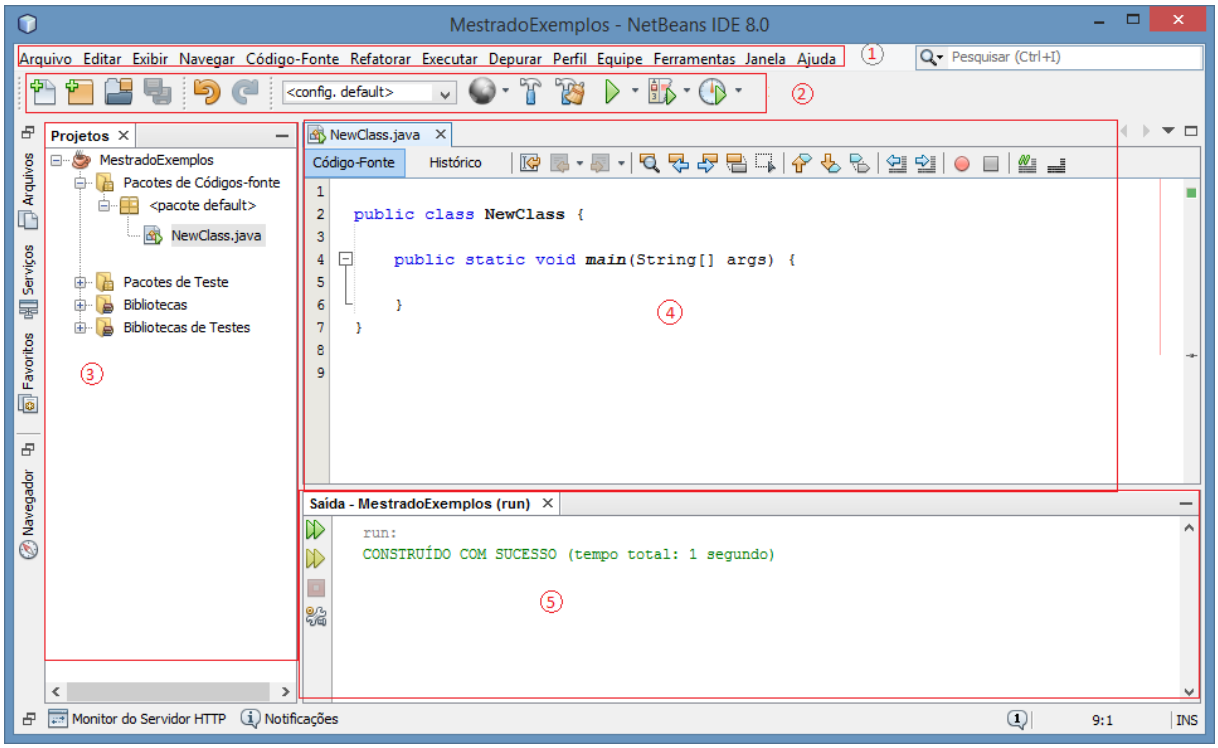


Figura 4.1: Janelas no Netbeans IDE.

Como destacado na Figura 4.1 verificamos a janela principal da ferramenta proposta para os trabalhos de desenvolvimento de código, por parte dos estudantes. Destacado em vermelho temos os principais itens de interação com o usuário:

(1) Barra de Menus - que oferece acesso a itens de configuração e manipulação da ferramenta;

(2) Barra de Ferramentas - apresenta alguns item de ação que são usados com maior frequência;

(3) Aba de Projetos - apresenta uma visão lógica do conteúdo do projeto;

(4) Editor de Código - onde a edição do código fonte para construção de programas é realizada;

(5) Aba de Saída - que mostra resultados de execução dos programas, erros de compilação e informações auxiliares.

O NetBeans foi adotado como ferramenta de desenvolvimento, pois a construção de uma ferramenta completa de desenvolvimento seria inviável e fugiria do escopo desse trabalho. Apesar da escolha, outras ferramentas de software podem se beneficiar dos conhecimentos apresentados aqui, bastando apenas dar enfoque nas particularidades de implementação dos requisitos que sustentam o auxílio a Composição de Resultado.

4.2 LabProg Web

Em função dos problemas relacionados ao gerenciamento e retrabalho dos professores e alunos em uma disciplina de programação, foi realizado um trabalho de infraestrutura. Assim como agregando abordagens presentes na literatura, visando facilitar os trabalhos de correção das atividades e a oferta de feedbacks de ajuda, que proporcionem o auxílio do professor aos estudantes.

Foi construído um sistema web denominado LabProg Web – Laboratório de Programação, para o professor gerenciar os estudantes e suas atividades, listas e problemas. Já os estudantes visualizam as atividades pendentes de resolução e acompanham o seu desempenho registrado pelo sistema. A página inicial do sistema é apresentada na Figura 4.2 .

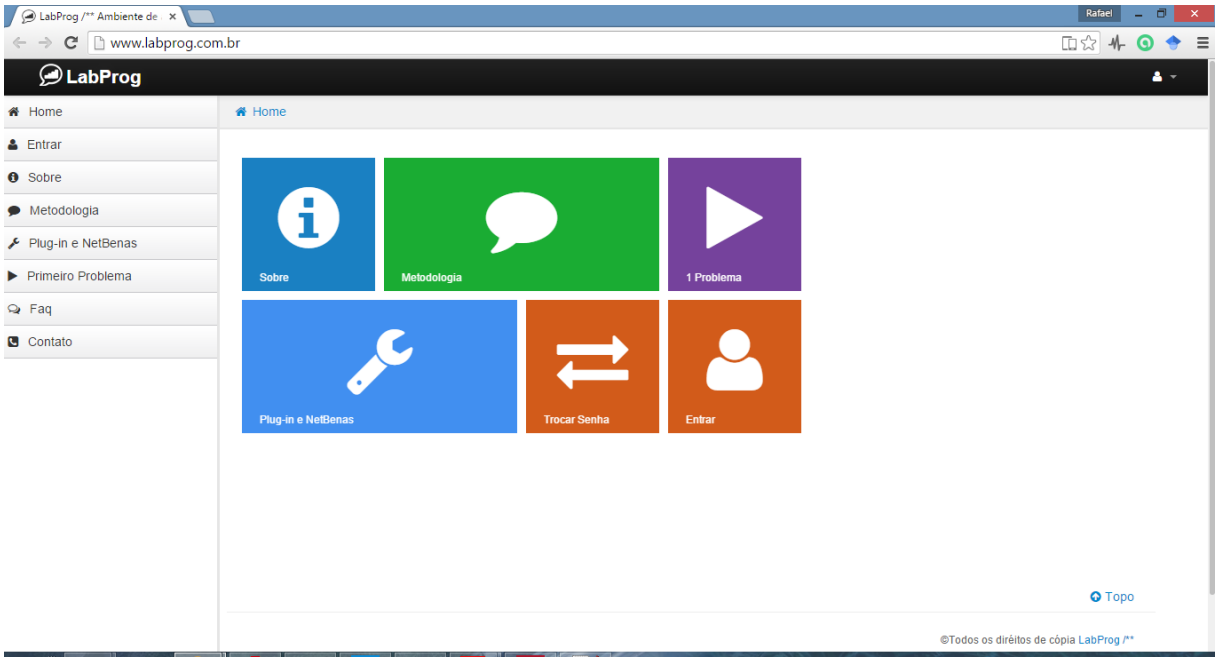


Figura 4.2: Página inicial LabProg.

O funcionamento das ferramentas, de modo geral, orienta e dá suporte a resolução de problemas. O ambiente LabProg proporciona o gerenciamento, controle e acompanhamento dos estudantes e problemas. Já o estudante pode verificar as atividades pendentes(problemas) e o seu desempenho.

O estudante trabalha no IDE, que por sua vez, obtém as informações do problema em segundo plano a partir do ambiente LabProg Web, como os testes e as sugestões para auxiliar a resolução. Caso a resposta do aluno passe em algum testes, o aluno é informado sobre o acerto, caso contrário ele apresenta uma sugestão, considerando a método de composição de resultado e qual teste falhou.

Todas as interações do estudante com o IDE de programação são registradas e enviadas para a ambiente LabProg Web, para compor estatísticas e registrar os erros e acertos.

Os testes isolam as entradas e as saídas dos subproblemas, onde cada teste possui foco especifico, sendo possível identificar qual passo do método o estudante se encontra, e assim, ofertar uma sugestão para resolução baseada no contexto atual.

A elaboração dos casos de teste pode demandar bastante tempo e esforço por parte do professor, entretanto este se justifica pela utilização futura dos testes e problemas. Os testes não são exibidos, eliminando a tentativa de fraude dos estudantes tentam produzir o resultado sem responder o problema.

4.2.1 Funcionalidades do LabProg

As funcionalidades do sistema web estão representadas na Figura 4.3. Elas estão focadas em três tipos básicos de usuários, sendo o professor, aluno e administrador. O tipo do usuário é definido por um perfil de segurança, ou seja, no cadastro de usuário ele pode receber qualquer perfil e suas devidas permissões.

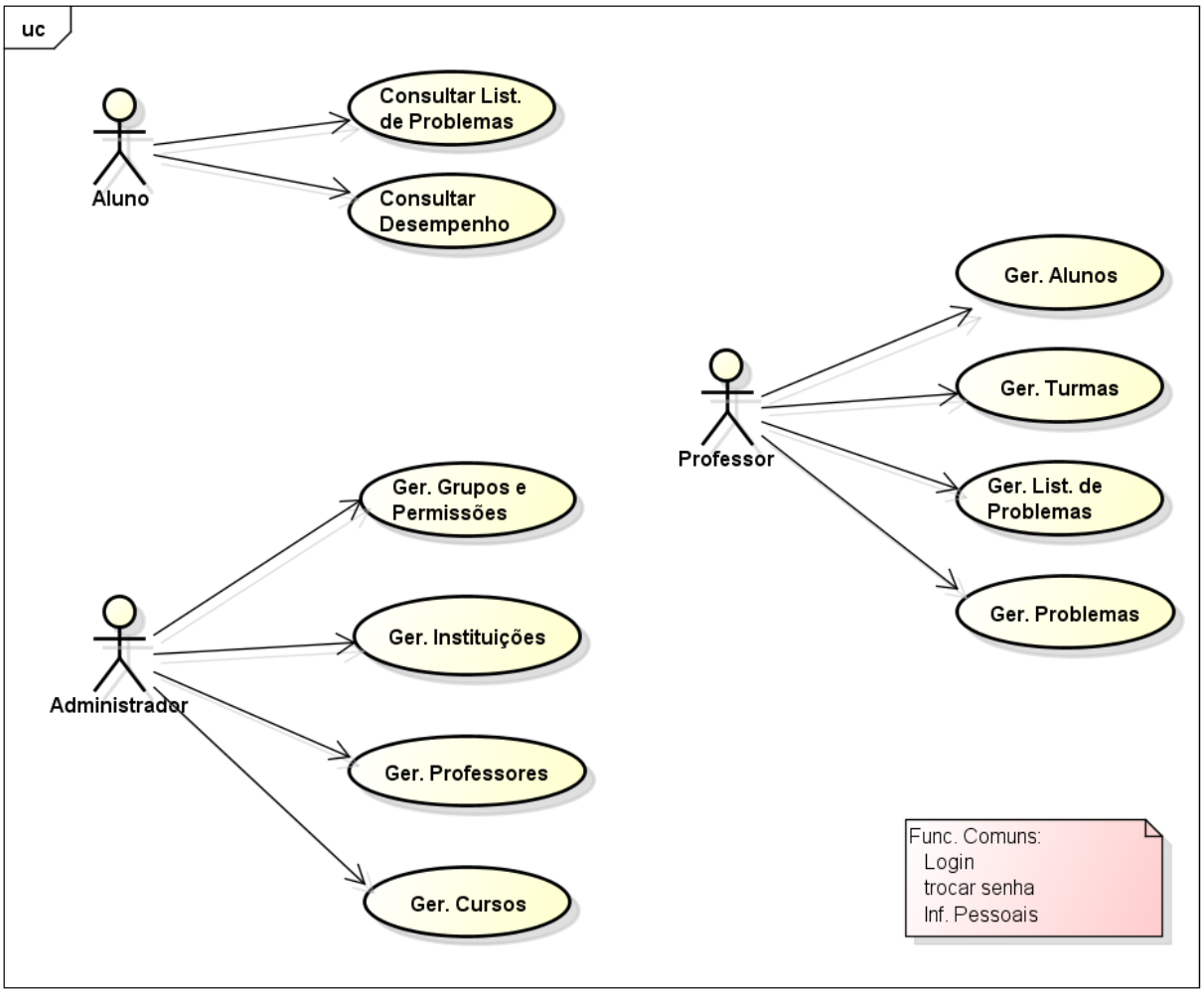


Figura 4.3: Diagrama de casos de uso das principais funcionalidades.

No sistema web o Aluno pode fazer o acompanhamento do seu desempenho e verificar

as suas pendências de resolução. Essas funcionalidades são:

Consultar lista de problemas - Essa atividade está disponível por meio da opção "Listas", nela são apresentadas todas as atividades que o estudante pode desenvolver; Um exemplo da visualização do estudante pode ser analisado na figura a seguir:

Figura 4.4: Listagem de um problema para o estudante.

Consultar desempenho - Esta funcionalidade apresenta todas as listas de problemas associadas a turma do estudante, com uma barra de progresso de suas atividades e o detalhamento de qual problema ainda está pendente.

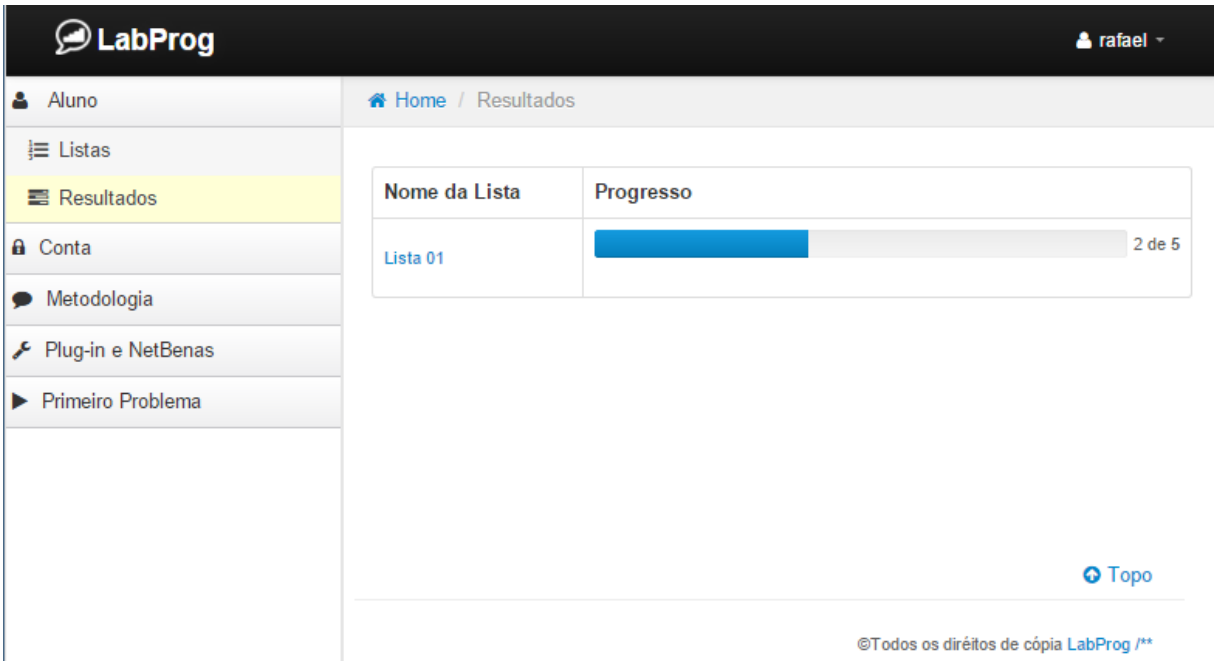


Figura 4.5: Progresso de resolução da Lista de Problemas.

O Professor desempenha as funções de gerenciamento dos principais itens do sistema web, sendo elas:

Gerenciar Turmas - Essa atividade é referente cadastro de Turmas, especificando informações do curso, professore e período de existência;

Gerenciar Alunos - Nesse item o professor manipula informações como nome, matrícula e associa os estudantes a usuários com perfis de segurança e turmas que está presente;

Gerenciar Problemas - Na elaboração de um problema o professor deve inserir a definição, exemplos de entrada e saída e informar os testes, que são entradas e saídas associadas para automatizar a correção. Neste ponto ele pode associar sugestões aos testes, para que em uma eventual falha esse auxílio possa ser apresentado. Temos uma ilustração do cadastro do problema na Figura 4.6;

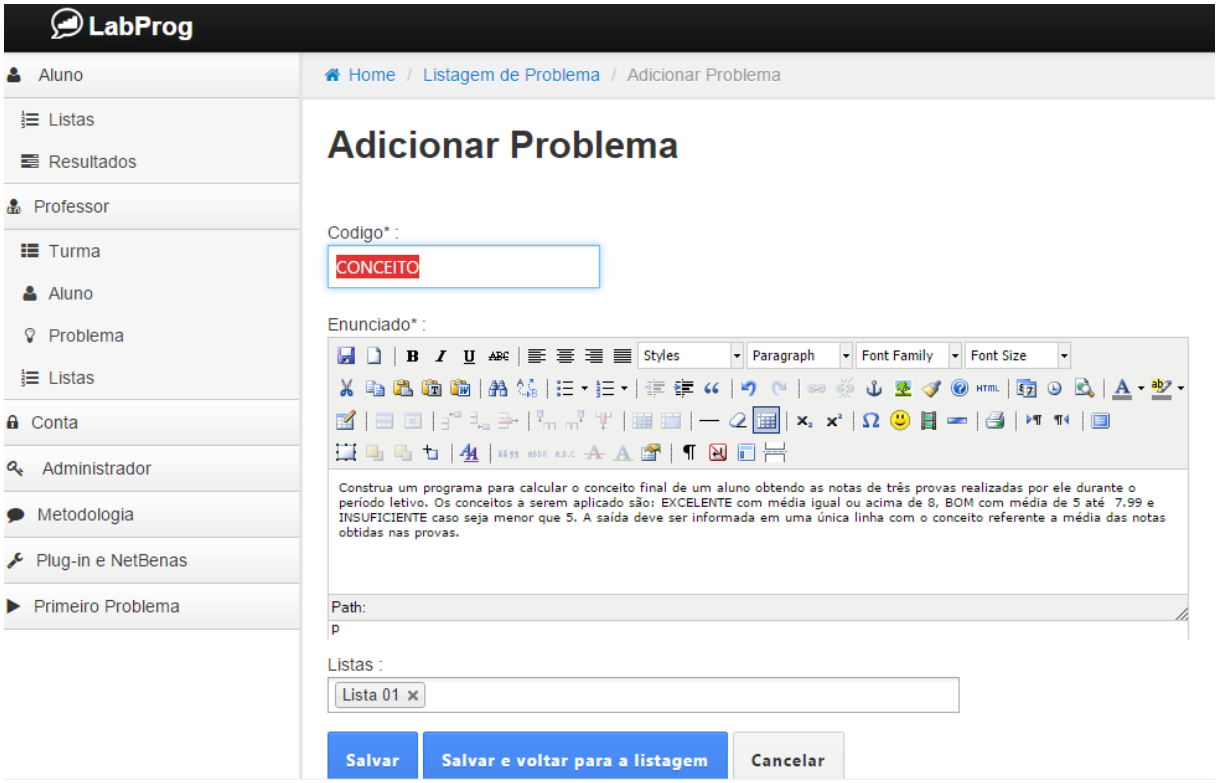


Figura 4.6: Tela de cadastro do Problema.

Gerenciamento de Teste - O gerenciamento dos testes deve ser realizada para cada problema. Cada problema deve ter um conjunto de cadastrados, como mostrado na Figura 4.7, deste modo, informando a entrada, a saída e a dica(Sugestão do Método) caso o teste falhe.

Figura 4.7: Tela de cadastro do Teste.

O atributo Ordem define uma hierarquia na ordem de execução dos testes, assim os testes executados de maneira sequencial e não avançam para um próxima em caso de falha. Vários testes podem ser cadastrados com uma mesma ordem, e na execução esses são selecionados de maneira aleatória para a mesma ordem.

Portanto, podemos fazer uma associação direta entre o atributo Ordem do teste com um determinado subproblema, assim os vários teste de uma ordem podem ter o foco de um determinado subproblema. Caso não haja necessidade de ordenar a sequência dos testes dos subproblemas, basta cadastrá-los com o mesmo número de ordem;

Gerenciar Listas de Problemas - Os problemas são agregados em listas de problemas, assim, na definição de uma lista são inseridas informações como o nome da lista,

período de aplicação, a turma que irá responder e o tipo que pode ser de Atividade ou Avaliação, sendo que neste último os sistema não apresenta informações auxiliares (dicas).

O perfil de usuário Administrador realiza todos os cadastros auxiliares, como as instituições de ensino, cursos, professores, grupos de usuários(perfis) e permissões. O sistema de segurança foi concebido e integrado para que ele funcione de modo dinâmico, assim permitindo que usuários e grupos novos sejam criados e modificados. Como na necessidade do professor ter um monitor basta cadastrar um novo perfil com as devidas permissões.

4.3 Plug-in LabProg

Como mencionado em seções anteriores, já temos o ambiente de programação Netbeans IDE para o desenvolvimento das atividades práticas de codificação. Juntamente com o sistema web LabProg que gerencia os recursos como alunos, listas e problemas. Entretanto, surge a necessidade de integração para que esses elementos se comuniquem.

Assim, foi concebida uma ferramenta de integração, o Plug-in LabProg. Uma extensão de funcionalidades do programa NetBeans, que possui entre suas principais funções obter informações dos problemas, assim como informar os erros e acertos dos estudantes.

As novas funcionalidades implementadas aos NetBeans foram associadas a um botão, como pode ser percebido na Figura 4.8 destacado com um círculo vermelho, além de ilustrar a tela inicial da ferramenta , menus e funcionalidades básicas do NetBeans IDE.

Figura 4.8: Botão do Plug-in LabProg no Netbeans IDE.

Este, quando acionado realiza a identificação do estudante, do problema que pretende resolver. Assim pode-se obter as informações do problema e informa os erros e acertos dos estudantes para registro. Podemos destacar e descrever as principais atividades que ele realiza que Plug-in Labprog executa:

Identificação de Usuário - Essa tarefa trata da identificação do estudante por meio de um usuário e senha, que são checados administrados no Labprog Web;

Identificação do Problema - A ferramenta tenta encontrar no Editor de Código uma janela ativa contendo o identificador do problema em qualquer comentário. Ex: //@problema NOME_PROBLEMA;

Compilação - É realizada uma compilação de todos os arquivos de código para garantir q o código enviado está com sintaxe valida e evitar problemas de cache de arquivos;

Obtenção de Informações - O identificador do problema é enviado para o Labprog Web, assim ele pode checar se o problema deve ser respondido pelo estudante, ou seja, se está associado a uma lista de problemas com período de submissões válido. Caso o resultado da checagem seja positivo as informações são envidas do Labprog Web para o IDE, como enunciado do problema, testes e dicas (feedbacks);

Testes - Com base nas informações obtidas, o programa feito pelo estudante é executado em segundo plano, e os testes são realizados.

Submissão - O código da resposta é enviado para registro no Labprog Web, um informe se está correto e a identificação do teste caso tenha falhado;

Feedback - No final dos testes o estudante é informado se a sua solução está correta ou em quantos testes ela falhou, além de ser ofertada uma sugestão baseada no teste que falhou. Essas dicas de auxílio são apesentadas na Aba de Saída do Netbeans, e um exemplo pode ser visualizada como na imagem que segue.

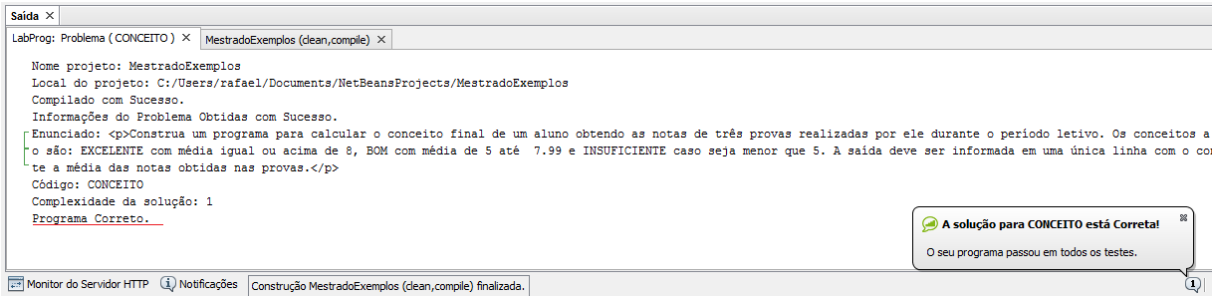


Figura 4.9: Feedbacks de auxílio(Dicas).

4.3.1 Testes Parciais

O plug-in apresenta como principais pontos de integração com o método da Composição de Resultado o teste parcial e a oferta de feedbacks.

O teste parcial é fundamental para que o estudante possa progredir na resolução de um dado problema, levando em consideração que pode verificar se parte da implementação que está produzindo apresenta erros e quanto do problema consegue responder.

Esse teste é focado em um dado subproblema, como discutido na Composição de Resultado, então a ideia básica é que o estudante possa realizar um teste com uma entrada que tente isolar o subproblema em questão. Assim, eliminando interferências causadas por outros subproblemas ainda não tratados.

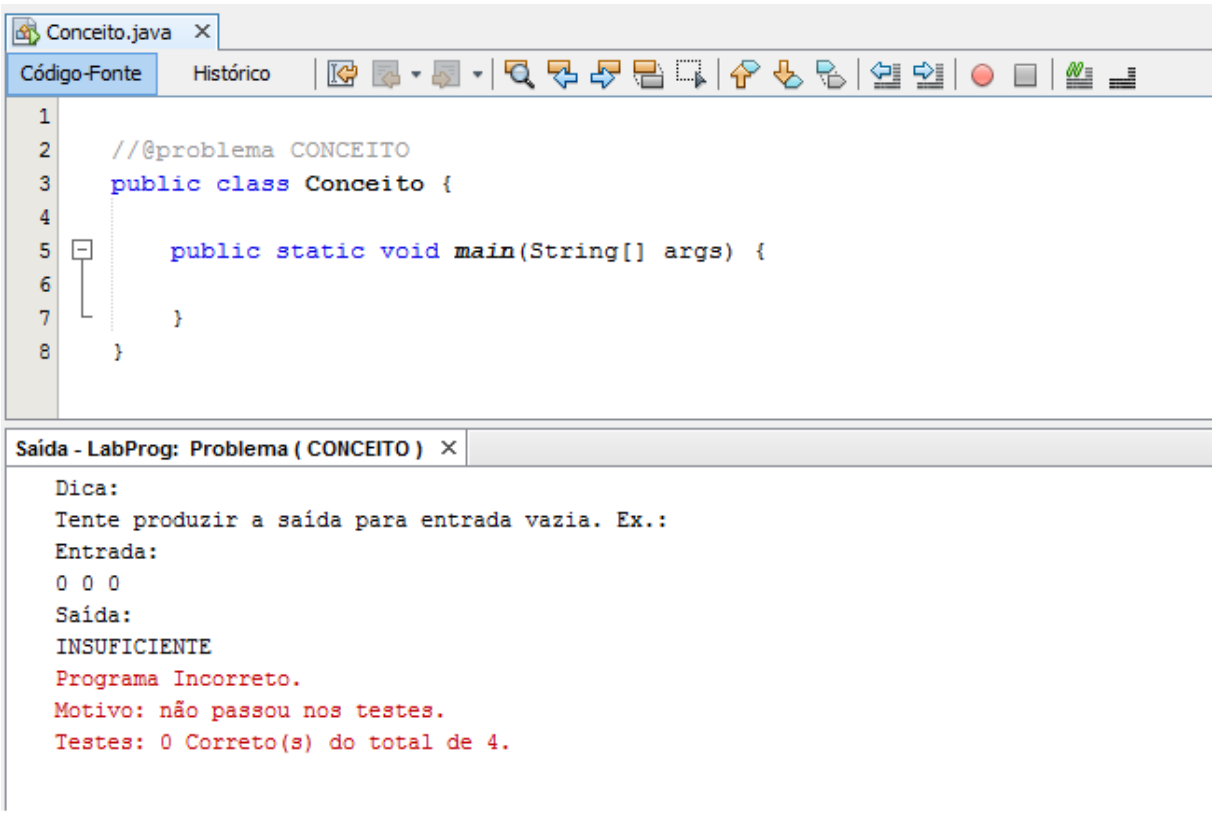


Figura 4.10: Teste parcial sem implementação.

A partir de um mínimo de código o programa já pode ser testado, como o exibido na Figura 4.10, onde é criada a base para um programa, mas sem nenhuma implementação. Ao ser pressionado o botão do Plug-in Labprog executa as tarefas descritas anteriormente e apresenta um feedback indicando que o estudante deve implementar a codificação um determinado subproblema exemplificado a o que poderia ser utilizado como entrada e saída.

Como exemplo da sequência dos testes e feedbacks apresentados pela ferramenta temos na Figura 4.11 a concepção da base de um programa para resolução de um problema. Apesar de trivial, já indica um possível início de resolução e um próximo subproblema que deve ser resolvido. A sequência de passos apresentadas no capítulo 3 para esse problema deve ser seguida até atingir como resultados final os informativos apresentados Figura 4.9.

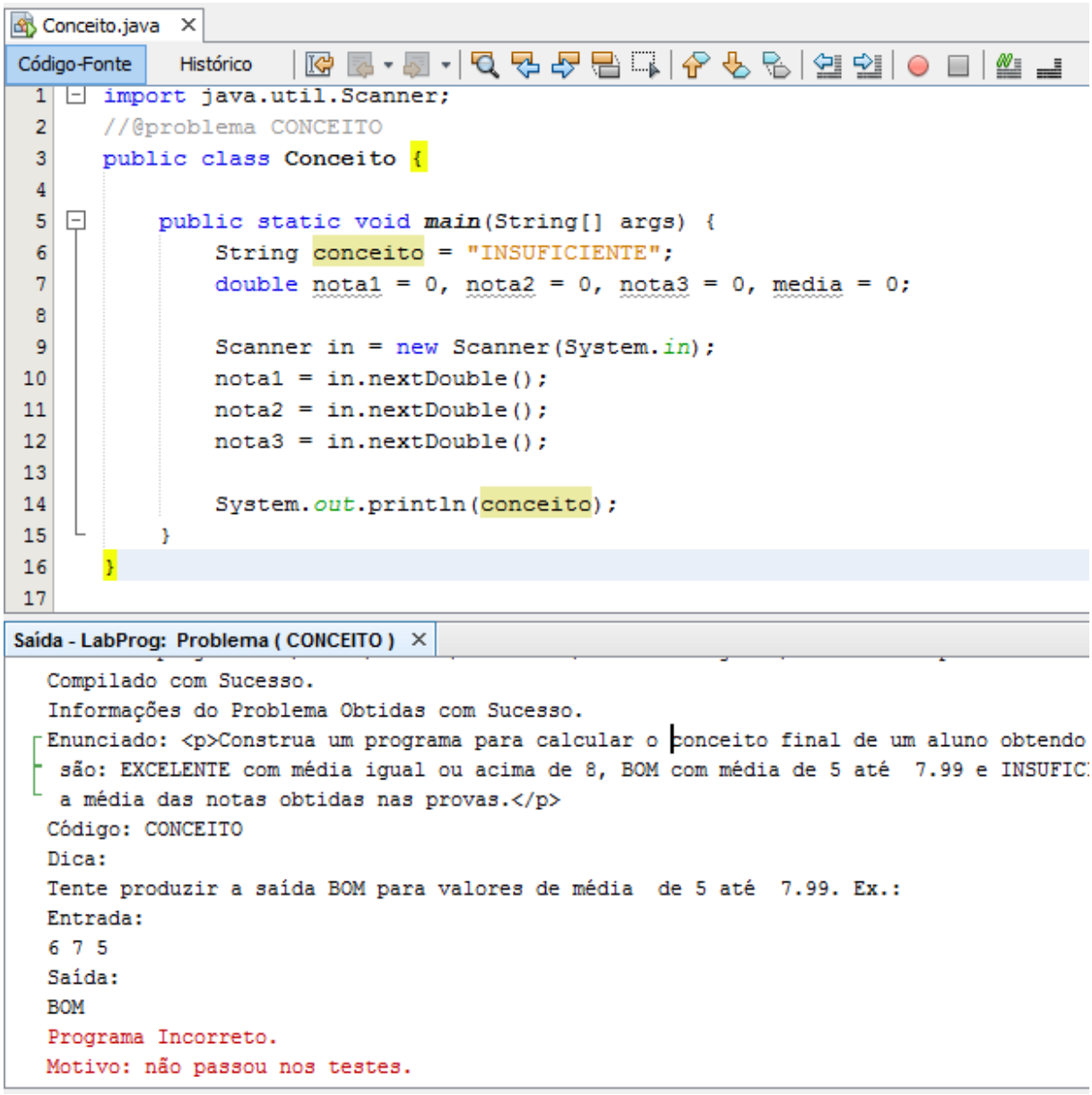


Figura 4.11: Teste parcial com implementação básica.

4.3.2 Feedback

Os feedbacks apresentados pela ferramenta são de dois tipos, os feedbacks gerados pela ferramenta que indicam informações da utilização da ferramenta e dados mais gerais de funcionamento, como podem ser observados na Figura 4.12. E os feedbacks do método que informam em qual passo o estudante está e que ações deve tomar para conduzir a construção da resposta.

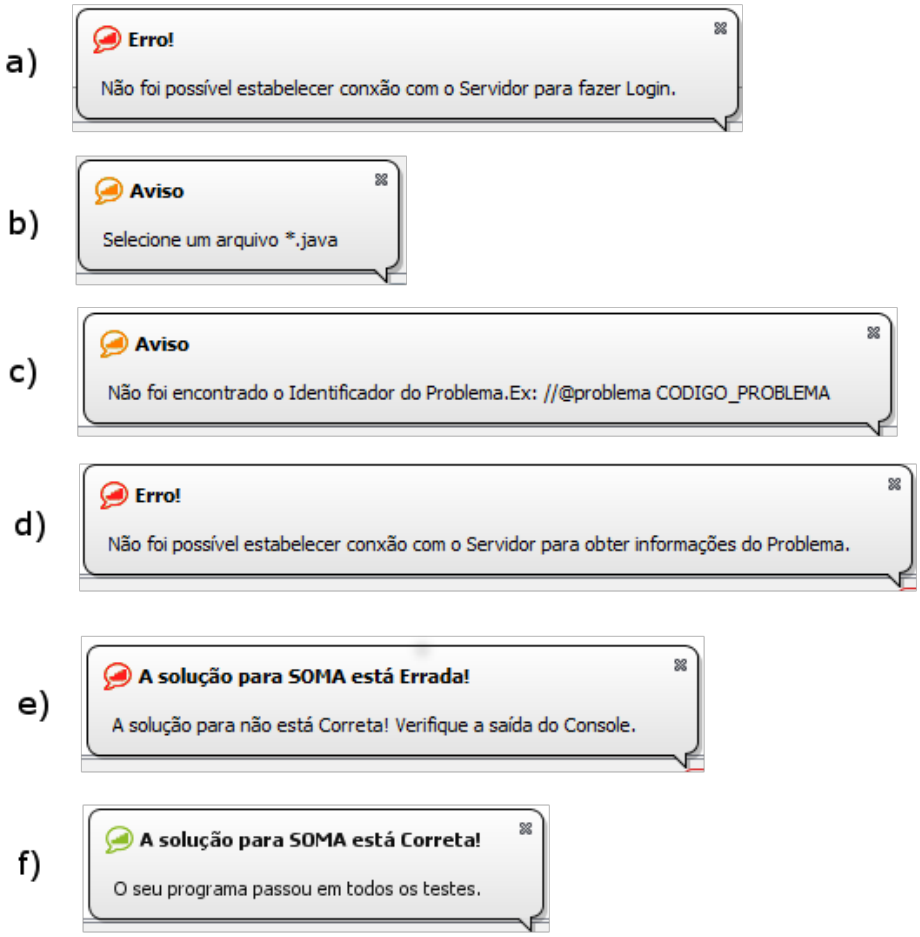


Figura 4.12: Feedbacks apresentados no funcionamento da ferramenta.

Os feedbacks mostrados na imagem são apresentados nas seguintes ocasiões: a) e d) são apresentados caso haja alguma falha de comunicação de rede. Os itens b) e c) são apresentados respectivamente, na situação de o estudante tentar submeter algum código que não seja em java, e na ausência da identificação do problema, necessária para saber informações do problema e qual o estudante está resolvendo. O item e) é apresentado nas ocorrências de erros do estudante(detalhados na saída do console) e f) quando consegue passar em todos os teste, assim obtendo uma solução correta.

Para exemplificar os feedbacks do método mostraremos uma sequência de figuras indicando como pode ser feito um cadastro completo dos testes com seus respectivos feedbacks(Dicas). Considerando problema de ler duas variáveis e efetuar um a soma delas, um problema de saída convergente.

Figura 4.13: Cadastro do primeiro teste para o problema Soma.

Na Figura 4.13 temos a apresentação do cadastro do primeiro teste, com Ordem de valor 1(um), ou seja, será o primeiro teste a ser executado. Caso ele falhe os teste de ordem maior não serão executados. A dica(feedback) será apresentada caso ele falhe, esse teste trabalha com a concepção da base do programa.

Figura 4.14: Cadastro do segundo teste para o problema Soma.

Na Figura 4.14 temos a apresentação do cadastro do segundo teste, com Ordem de valor 2(dois). Trata de um subproblema específico, a implementação da soma considerando que ela é gerada pela primeira variável.

Figura 4.15: Cadastro do terceiro teste para o problema Soma.

Na Figura 4.15 temos a apresentação do cadastro do terceiro teste, com Ordem de valor 2(dois). Trata de um subproblema específico, a implementação da soma considerando que ela é gerada pela segunda variável. Esse teste possui a mesma ordem do aprestado na Figura 4.14, portanto, no momento da execução dos testes, eles serão executados com ordem aleatória.

Figura 4.16: Cadastro do quarto teste para o problema Soma.

Na Figura 4.16 temos a apresentação do cadastro do quarto teste, com Ordem de valor 3(três). Trata do teste de todos os subproblemas em conjunto, ou seja, teste com o primeiro e segundo número para gerar a soma. Esse tipo de teste indica se há a necessidade de fazer alguma alteração para que a resposta dos subproblemas funcionem em conjunto.

4.4 Restrições

Apesar do Ambiente de Aprendizagem ter por objetivo ofertar facilitadores para a implementação do método da Composição de Resultado, ele apresenta algumas restrições provenientes das escolhas tecnológicas de implementação e características da sua utilização.

4.4.1 Internet

O Netbeans IDE funciona a partir das informações obtidas do Labprog Web, que é um sistema web e funciona online, deste modo é essencial possuir conexão de Internet para o uso das ferramentas. Entretanto, essa restrição é muito importante para o acompanhamento em tempo real do desempenho dos estudantes. Assim como realizar correções das atividades, mesmo depois dos estudantes já terem iniciado a resolução. Contudo, pode ser usado em uma estrutura de rede local.

4.4.2 Versão

Para este estudo foi utilizado o Netbeans IDE na versão 8.0. Deste modo, o plug-in foi construído com foco nessa versão e apesar dele funcionar em outras recomenda-se o uso desta, pois nas mudanças de versão do Netbeans IDE ele atualiza diversas bibliotecas internas. E algumas que possuem dependência com o plug-in, assim podendo apresentar mal funcionamento. Para resolução deste possível problema, basta realizar atualizações de acordo com as mudanças do Netbeans, quando necessário.

4.4.3 Saída

Como mencionado anteriormente no funcionamento dos testes, o resultado produzido pelo programa é comparado com o desejado para um dado subproblema. Entretanto, os estudantes e professores de programação fazem uso de mensagens auxiliares como: "Informe o primeiro número". Devido a esta "interferência" na saída do programa, foi criado um mecanismo para ignorar essas mensagens, deste modo toda mensagem que inicia por # é ignorado como resultado da execução do programa.

4.5 Considerações

4.5.1 Avaliações e Competições

O ambiente de ensino pode ser usado para atribuir notas, sendo que atualmente essa avaliação fica a cargo do professor. Mas, futuramente pode ser implementado um mecanismo que atribua notas baseados nos acertos dos problemas, podendo ainda basear-se no acerto parcial que o método proporciona.

Existe a possibilidade de realizar competições e avaliações utilizando o ambiente educacional proposto, visto que as listas de problemas podem ter um foco avaliativo, e não fornecem os feedbacks do método quando configuradas desta maneira. Assim, é esperado que com o tempo a atividade de pensar nos teste venha a facilitar a divisão dos subproblemas.

4.5.2 Dependência da tecnologia

Como destacado por Wolz (2001), o feedback imediato pode deixar os estudante muito dependentes da tecnologia, para tanto, desenvolvemos esse trabalho dando prioridade por usar uma ferramenta já estabelecida, que o estudante possa continuar usando futuramente, o NetBeans IDE. Assim, tentando evitar que o estudante só saiba resolver problemas utilizando um ambiente de ensino específico.

Colaborando com essa ideia, podemos notar no exemplo do cadastro completo dos testes e feedbacks de um problema, e foi dada prioridade por mostrar em qual passo o estudante está e um teste para que ele possa pensar em um determinado subproblema. Portanto, não dizemos qual subproblema deve resolver e sim um possível teste para lhe orientar sobre o subproblema.

4.5.3 Diferenciais para outras Abordagens

Um ponto importante é o funcionamento em tempo real das ferramentas, pois o educador pode identificar quais estudantes necessitam de mais auxílio, quais questões apresentam mais respostas com erro. Assim podendo providenciar intervenções como foco específico. Para o estudante é essencial obter informações sobre a correção da sua resposta, além de auxílio para progredir(dicas).

Apesar da correção ser baseada em entradas e saídas como em diversos outros trabalhos, ela se mostra flexível, pois o método preconiza a ideia dos testes parciais. Assim considerando que parte da resposta pode estar certa, e orienta se após as modificações os itens anteriores continuam corretos.

A criação dos programas que respondem os problemas não necessitam de formalismos no sentido de determinar o nome do programa ou variáveis, bastando informar o identificador em um comentário. Assim, deixando o estudante livre para desenvolver sua solução sem limitadores criativos, bastando que sua solução execute e produza os valores corretos.

Como o intuito de prover um indicativo de "qualidade" foi inserido no feedback um informe com valor da complexidade Ciclomatica (MCCABE, 1976) quando uma resposta é considerada correta, para que o estudante seja incentivado a melhor seu código(refatorar) com base em uma métrica. Este indicativo foi inserido em caráter experimental e como forma de exemplificar indicadores que poderão ser inseridos para análise e comparação de código.

Capítulo 5

Resultados e Discussão

Como forma de avaliar a viabilidade do uso prático do método de composição de resultado aliado ao ambiente de auxílio LabProg, foram concebidos experimentos para avaliar a aderência deles ao proposto como objetivo nesse trabalho.

5.1 Experimento Prático com Alunos

Os estudantes foram divididos em dois grupos aleatoriamente, o grupo de Controle formado por 19 participantes, e o grupo Experimental com 20 integrantes que utilizaram o método e as ferramentas. Ambos os grupos possuíam condições semelhantes em relação aos professores, laboratório e tempo de execução nas atividades. O delineamento experimental foi inspirado no trabalho de Queirós e Leal (2012).

As sessões de práticas de laboratório se deram em 7 encontros de uma hora e meia. Desta forma, em cada um foi apresentado uma Lista de 4 problemas que totalizam 28 problemas desenvolvidos em 10,5 horas de atividades. Os encontros em ambos os grupos, contavam com uma breve revisão de conhecimentos teóricos de programação e seguido pela execução de uma lista de problemas. No último encontro eles responderam um questionário como perguntas relativas ao uso e satisfação do método e dos sistemas.

Durante a condução dos encontros com os estudantes foi adotada uma política de intervenção mínima, ou seja, quando os estudantes solicitavam ajuda não foram dadas informações acerca da resposta. Entretanto, foram questionados sobre o que não haviam entendido. Na ocorrência de erros sintáticos eles foram orientados a identificar a linha que apresentava erro e analisar a informação de erro que o IDE produz.

5.1.1 Preparação do Experimento prático

A preparação do experimento foi focada na seleção de problemas, que indicassem as possibilidades de problemas que podem ser trabalhados pelo método. Além de favorecer as análises decorrentes da Composição de Resultado.

Um dos fatores mais importantes para escolha do problema foi o quanto os alunos poderiam se beneficiar com os itens trabalhados na resolução de um determinado problema. Portanto, foram feitas muitas analises para averiguar quais características seriam interessantes para que os problemas contivessem, como uma categorização, nível de dificuldade e raciocínios comuns entre os problemas.

A distribuição dos problemas em categorias foi inspirado no trabalho de Rocha et al. (2010), que trabalha o ensino de programação baseando-se em níveis relacionados aos conceitos teóricos. Assim temos as seguintes categorias:

Introdução problemas que apresentam cálculos matemáticos simples, declaração e manipulação de variáveis, assim como comandos de entrada e saída;

Seleção Simples problemas que devem ser resolvidos com uma instrução de seleção simples(instrução if). Executa uma ação se uma condição é verdadeira e pula-a, se falsa;

Seleção Composta problemas que podem ser resolvidos com uma estrutura de seleção dupla/composta (instrução if..else), que realiza uma ação se uma condição for verdadeira e uma ação diferente se a condição for falsa. Podendo ser agrupada, como no caso das instruções IF aninhadas;

Repetição while problemas que apresentam comportamentos iterativo, podendo ser controlada por flag ou contador, sendo que ela pode resolver qualquer problema de repetição;

Repetição do..while problemas de repetição semelhantes aos da repetição while, contudo diferindo pela estrutura executar o seu bloco de instruções ao menos uma vez e seu teste de continuidade ocorrendo no final da estrutura;

Repetição for problemas com comportamento repetitivo, entretanto usado repetições controladas por contador, pois são laços bem definidos quanto a quantidade de iterações.

Os problemas também foram classificados com um grau de dificuldade, considerando uma análise dos conhecimentos necessários para resolve-lo. Como conhecimentos relacionados a estruturas de programação, elementos sintáticos e uso de lógica matemática.

Nível 1 - Baixa dificuldade;

Nível 2 - Média dificuldade;

Nível 3 - Alta dificuldade.

Tabela 5.1: Organização dos problemas nas listas.

Ord.	Lista	Problema	Categoria	Dificuldade
1	Lista 1	SOMA	Introdução	Baixa
2	Lista 1	CALIBRANDO_PNEUS	Introdução	Baixa
3	Lista 1	MEDIA_ALUNO	Introdução	Baixa
4	Lista 1	METADE_MAIOR_QUE_20	Seleção Simples	Baixa
5	Lista 2	APROVADO	Seleção Simples	Média
6	Lista 2	RAIZ_DO_NEGATIVO	Seleção Simples	Média
7	Lista 2	TRIPLO_DO_MAIOR	Seleção Composta	Alta
8	Lista 2	JOGO_PAR_OU_IMPAR	Introdução	Alta
9	Lista 3	APROVADO_OU_REPROVADO/AP	Seleção Composta	Baixa
10	Lista 3	IMC	Seleção Simples	Média
11	Lista 3	TOMADAS	Introdução	Alta
12	Lista 3	VICE_LIDER	Seleção Composta	Alta
13	Lista 4	CONTROLE_DE_QUALIDADE	Seleção Composta	Baixa
14	Lista 4	DESAFIO_DA_MAIOR_PALAVRA	Seleção Composta	Média
15	Lista 4	FLIPER	Seleção Composta	Baixa
16	Lista 4	A_MAIS_PEDIDA_DA_SEMANA	Repetição for	Alta
17	Lista 5	DIVISORES_DE_N	Repetição while	Alta
18	Lista 5	ZERINHO_OU_UM	Seleção Composta	Média
19	Lista 5	DESARIO_DO_MAIOR_NUMERO	Repetição do..while	Alta
20	Lista 5	NOTAS_DA_PROVA/NOTAS_D	Seleção Composta	Média
21	Lista 6	CONTA_DE_AGUA	Seleção Composta	Alta
22	Lista 6	AUDIENCIA	Repetição for	Alta
23	Lista 6	SALDO_DA_VOVO	Repetição while	Alta
24	Lista 6	MEDIA_TURMA_10_ALUNOS	Repetição for	Alta
25	Lista 7	MEDIA_TURMA	Repetição do..while	Alta
26	Lista 7	RECEITA_DE_BOLO	Seleção Composta	Alta
27	Lista 7	OBI	Repetição while	Alta
28	Lista 7	FIBONACCI	Repetição for	Alta

Os problemas podem ser consultados no Apêndice E - Problemas das listas de Programação, entretanto na Tabela 5.1 é apresentada a distribuição dos problemas nas sete listas, além de apresentar a categorização e nível de dificuldade.

Na distribuição dos problemas optou-se por distribui-los considerando misturar os elementos de categorização, além dar prioridade pela ascendência da dificuldade e complexidade das estrutura necessárias.

A distribuição dos acertos por questão é exibindo na Tabela 5.2, onde são mostradas as quantidades de acertos considerando cada problema e grupo do experimento.

Tabela 5.2: Acertos dos estudantes nos grupos.

Ord.	**Problema**	**Controle**	**Experimental**
1	SOMA	12	17
2	CALIBRANDO_PNEUS	7	14
3	MEDIA_ALUNO	7	14
4	METADE_MAIOR_QUE_20	9	13
5	APROVADO	8	13
6	RAIZ_DO_NEGATIVO	8	9
7	TRIPLO_DO_MAIOR	5	7
8	JOGO_PAR_OU_IMPAR	2	6
9	APROVADO_OU_REPROVADO/AP	11	17
10	IMC	11	14
11	TOMADAS	6	12
12	VICE_LIDER	5	10
13	CONTROLE_DE_QUALIDADE	9	14
14	DESAFIO_DA_MAIOR_PALAVRA	5	12
15	FLIPER	6	13
16	A_MAIS_PEDIDA_DA_SEMANA	2	12
17	DIVISORES_DE_N	8	8
18	ZERINHO_OU_UM	13	13
19	DESARIO_DO_MAIOR_NUMERO	3	3
20	NOTAS_DA_PROVA/NOTAS_D	5	8
21	CONTA_DE_AGUA	5	6
22	AUDIENCIA	10	12
23	SALDO_DA_VOVO	8	10
24	MEDIA_TURMA_10_ALUNOS	9	15
25	MEDIA_TURMA	2	4
26	RECEITA_DE_BOLO	3	4
27	OBI	9	11
28	FIBONACCI	4	4

A distribuição dos acertos considerando a categoria é exibindo na Tabela 5.3.

Tabela 5.3: Acertos por categoria.

Categoria	**Controle**	**Experimental**
Introdução	34	63
Seleção Simples	36	49
Seleção Composta	67	104
Repetição while	25	29
Repetição do..while	5	7
Repetição for	25	43

A distribuição dos acertos por dificuldade é exibindo na Tabela 5.4.

Tabela 5.4: Acertos por dificuldade.

Dificuldade	**Controle**	**Experimental**
Baixa	61	102
Média	50	69
Alta	81	124

5.1.2 Análise de Desempenho

A averiguação de melhoria de desempenho foi realizada por meio da análise das médias de problemas resolvidos. Entretanto, como os grupos possuem quantidade de participantes diferentes, optou-se por analisar os acertos dos indivíduos nos grupos com o total de problemas que eles deveriam resolver.

Desta forma, como temos 28 problemas e grupos com 19 e 20 participantes, temos que o grupo experimental conta com uma amostra de tamanho de 560 elementos no total, já o grupo de controle conta com 532 medições.

De posse da média de acerto de cada grupo temos que avaliar a significância estatística entre as duas amostras analisadas. Nesse contexto, iremos aplicar um teste de comparação de médias para duas amostras independentes, para dados não pareados, ou seja, grupos distintos.

Adicionalmente, para a escolha do teste estatístico foi necessário analisar a normalidade da distribuição das amostras, para esse fim usamos o teste de normalidade Shapiro–Wilk (ROYSTON, 1982) e encontramos um p-valor < 0.05, assim rejeitando a hipótese que nossos dados fossem oriundos de uma distribuição normal.

A tarefa de mensurar diferença entre as médias será desempenhada por um teste estatístico, considerado a distribuição, não paramétrico que trabalha com dois grupos independentes. O teste de Mann-Whitney (MANN; WHITNEY, 1947) é adotado como alternativa ao Teste t, assim poderemos avaliar a significância das diferenças de médias dos grupos.

Para fazermos a avaliação fundamental deste trabalho temos que considerar como hipótese nula H_0 como sendo as médias dos grupos experimental e de controle sendo iguais, e H_1 as médias dos grupos experimental e de controle sendo diferentes, considerando um nível de significância de 5%. Temos na Tabela 5.5 o resultado da execução do Mann-Whitney:

Tabela 5.5: Comparação das médias de acerto.

Grupo Experimental			Grupo de Controle			Mann-Whitney
n	M	Desvio Padrão	n	M	Desvio Padrão	p-valor
560	**0,52**	0.480	532	**0,36**	0.499	0,000035

Observando ($p-valor = 0,000035 \implies p-valor < 0,05$) rejeitamos a hipótese nula H_0 e podemos concluir com 95% de confiança (ou uma chance de erro de 5%) que existe uma diferença estatisticamente significante entre as médias de acertos entre os grupos experimental e de controle.

Considerando a hipótese deste trabalho:

Propor um método de resolução de problemas de programação que oriente a construção do algoritmo, promovendo melhoria no desempenho da aprendizagem do estudante em atividades práticas.

Assim temos evidências científicas de que o desempenho do grupo que usou o método e as ferramentas teve um desempenho de aprendizagem melhor, deste modo validando positivamente a hipótese deste trabalho. E verificando as devidas proporções de erros e acertos temos uma diferença de cerca de 17% de acertos, a maior para o grupo experimental.

5.2 Apreciação dos Professores

Dos encontros, quatro foram realizados de maneira presencial e três por vídeo conferencia, assim os conceitos foram apesentados e eles utilizaram as ferramentas com acompanhamento, entretanto visando interferir o mínimo na experiência. Ao final do encontro responderam um questionário acerca da apreciação.

Para analisar o resultado dos questionários foi utilizada a distribuição por Frequência Absoluta, para apresentar os dados e suas respectivas frequências que ocorreram. Para cada pergunta é exibida a frequência das respostas dadas pelos participantes, de acordo com a escala de Likert (1932). Utilizou-se um valor de 1 a 5 para associados as categorias da escala usada, como está organizado na Tabela 5.6.

Tabela 5.6: Valores de Categorias

Valores Associados as Categorias					
Nenhum	Pouco	Moderado	Muito	Extremo	Média da Pergunta
1	2	3	4	5	$\bar{x}_{pergunta}$

Consideramos quanto mais próximos de 5(cinco), maior será o nível de concordância dos indivíduos em relação ao questionamento da pergunta. E consequentemente, um valor próximo de 0(zero) vai indicar discordância total com o item proposto.

O calculado do valor médio da avaliação de cada resposta baseando-se na Média Aritmética Ponderada, de modo a estimar um grau de concordância dos professores em relação às perguntas usadas para validação. Fazendo usa da formula que segue:

$$\bar{x}_{pergunta} = \frac{\sum_{i=1}^{n} x_i.f_i}{n}$$

Onde temos:

$\bar{x}_{pergunta}$ = Valor médio da avaliação das perguntas.
n = número de pessoas que responderam as perguntas.
x_i = valor associado à categoria da escala, Tabela 5.6.
f_i = quantidade de ocorrências de uma resposta(frequência).

Apresentamos as perguntas utilizadas no questionário respondido pelos professores para avaliar a sua apreciação do método e das ferramentas:

#P1 Como avalia o grau de **dificuldade** para o entendimento do método da Composição de Resultado para resolver problemas de programação?

#P2 Como avalia a **dificuldade** de instalação/uso das ferramentas de software apresentadas nesse (LabProg Web e Plug-in p/ Netbeans)?

#P3 Como avalia o grau de **importância** dos feedbacks que foram apresentados para a resolução dos problemas de programação, considerando iniciar e progredir a construção das respostas?

#P4 Como avalia o grau de **importância** do indicativo que demonstra se o programa está correto ou quantos testes restam para que a solução seja considerada completa?

#P5 Considerando o método e as ferramentas, como avalia o grau de **melhoria** pode ser ofertado em atividades práticas de programação?

#P6 Considerando o método e as ferramentas, qual grau de **interesse** teria em fazer uso em um curso de programação?

Tabela 5.7: Resultados da Apreciação do Professor

	Frequência Absoluta					
Pergunta	Nenhum	Pouco	Moderado	Muito	Extremo	$\bar{x}_{pergunta}$
#P1 - Dificuldade método	1	1	3	2	0	2,86
#P2 - Dificuldade instalação	5	2	0	0	0	1,29
#P3 - Importância feedback	0	0	1	4	2	4,14
#P4 - Importância correção	0	0	1	2	4	4,43
#P5 - Melhoria na aprendizagem	0	0	1	4	2	4,14
#P6 - Interesse em usar	0	0	1	3	3	4,29

O objetivo da Pergunta #P1 era ter elementos para mensurar um grau de dificuldade de entendimento do método, apesar do valor obtido não estar próximo de cinco, que indicaria um altíssimo grau de dificuldade. Infelizmente, temos predominância em categorias da escala que indicam considerado grau de dificuldade de entendimento. Como pode ser observado na Figura 5.1, as respostas estão concentradas nas categorias moderada e muita dificuldade. Esse fato indica que o método necessita de ajustes focados na apresentação, de forma a promover redução da dificuldade no entendimento do método da Composição de Resultado.

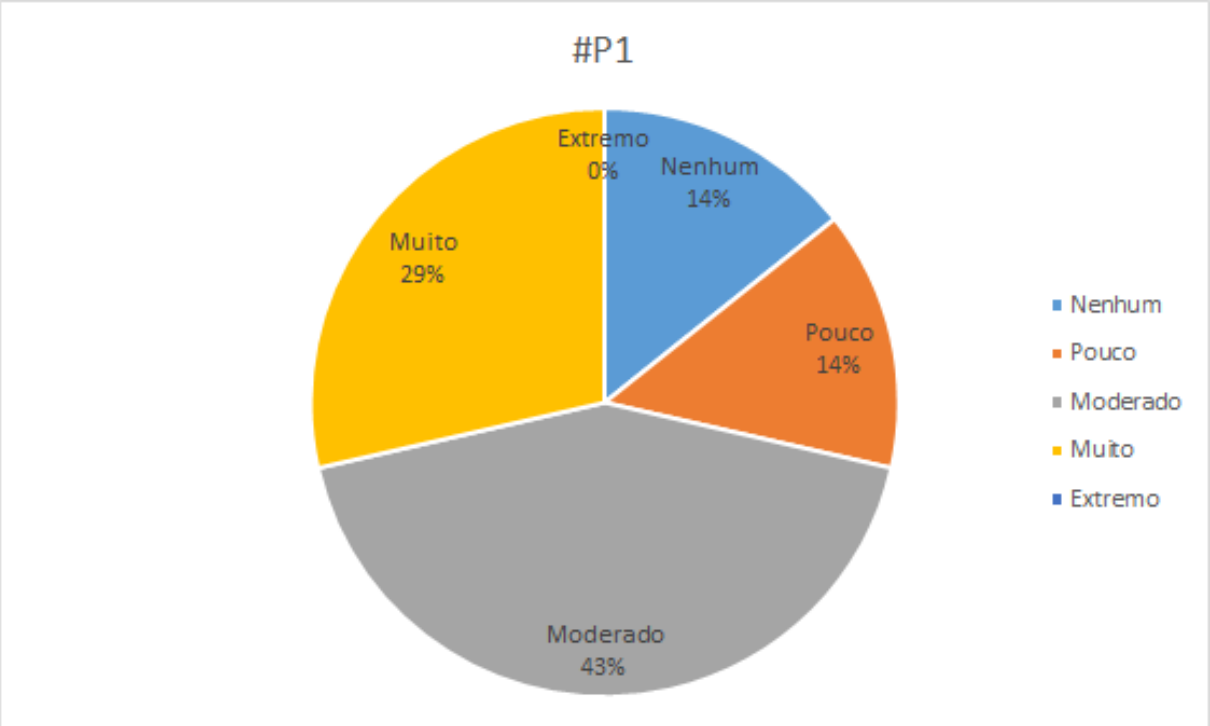


Figura 5.1: Distribuição gráfica das respostas da pergunta #P1.

A média da Pergunta #P2 indica que consideram a instalação e uso das ferramentas uma tarefa com baixa dificuldade, isso se deve ao fato de que a instalação do Netbeans

IDE é simples, pois bastam algumas confirmações nas telas de instalação. Já a instalação do plug-in é feita com poucos paços e o ambiente LabProg Web conta com manuais de instalação.

O grau de concordância das Perguntas #P3, #P4, #P5 e #P6 apresentam altos níveis de concordância. Portanto, temos uma apreciação muito positiva no que se refere aos feedbacks do método, nos indicativos de erros e acerto, nas possibilidades de melhorias e no interesse de utilização.

Após a responderem o questionário, dois professores indicarem que apesar do interesse não concordavam com a viabilidade de uso em suas realidades por estarem focadas no ensino introdutório de programação. A eles foi respondido que a partir dos bons resultados apresentados nesse trabalho, poderiam ser desenvolvidos ajustes nas ferramentas para trabalhos introdutórios de programação.

5.3 Satisfação dos Alunos

No último encontro do experimento prático de programação, os estudantes responderam um questionário para avaliarmos a satisfação destes em relação ao que foi exposto no decorrer dos encontros.

Apresentamos as perguntas utilizadas no questionário respondido pelos estudantes que avaliaram diversos itens relacionados ao método e as ferramentas:

#A1 Como avalia o seu conhecimento teórico sobre os conceitos de programação?

#A2 Como avalia o seu grau de entendimento do enunciado dos problemas de programação e a aplicação das estruturas de programação na resolução dele?

#A3 Como avalia a sua dificuldade para o entendimento do método da Composição de Resultado para resolver problemas de programação?

#A4 Como avalia a dificuldade de instalação/uso das ferramentas de software apresentadas nesse curso (LabProg Web e Plug-in p/ Netbeans)?

#A5 Como avalia o grau de importância dos feedbacks que foram apresentados para a resolução dos problemas de programação, considerando iniciar e progredir a construção das respostas?

#A6 Como avalia o grau de importância do indicativo que demonstra se o programa está correto ou quantos testes restam para que a solução seja considerada completa?

#A7 Considerando o método e as ferramentas, como avalia o seu grau de melhoria nas atividades práticas de programação?

#A8 Considerando o método e as ferramentas, você teria interesse em continuar fazendo uso para treinar suas habilidades de resolução de problemas?

Tabela 5.8: Resultados da Satisfação do Aluno

	Frequência Absoluta					
Pergunta	Nenhum	Pouco	Moderado	Muito	Extremo	$\bar{x}_{pergunta}$
#A1 - Grau de conhecimento	1	4	7	8	0	3,10
#A2 - Aplicação das estruturas	3	8	5	3	1	2,55
#A3 - Dificuldade método	1	2	7	8	2	3,40
#A4 - Dificuldade instalação	6	8	3	2	1	2,20
#A5 - Importância feedback	1	3	3	9	4	3,60
#A6 - Importância correção	1	1	2	6	10	4,15
#A7 - Melhoria na aprendizagem	0	3	6	9	2	3,50
#A8 - Interesse reúso	1	2	3	9	5	3,75

As perguntas #A1 e #A2 estão muito próximas do encontrado na literatura, pois os estudantes até conseguem entender os conceitos teóricos, entretanto apresentam muita dificuldade no emprego deles na resolução de problemas de programação.

A pergunta #A3 segue o indicado pelos professores de programação, pois os estudantes encontram um nível de dificuldade elevado para o entendimento do método, e que as propostas de melhorias são essenciais.

A dificuldade de instalação uso(#A4) das ferramentas pode ser considerado moderada conforme análise da sua média. A importância dos feedbacks são avaliadas positivamente, entretanto é dado um destaque acentuado nos relacionados a correção automatizada do código(#A6).

Eles indicaram estarem razoavelmente satisfeitos com o grau de melhoria obtidos (#A7). E também um considerável interesse em reutilizar o método e as ferramentas, em oportunidades futuras (#A8).

Capítulo 6

Conclusões

Para realizar a validação do objetivo principal deste trabalhos, consideramos a seguinte hipótese:

Propor um método de resolução de problemas de programação que oriente a construção do algoritmo, promovendo melhoria no desempenho da aprendizagem do estudante em atividades práticas.

O experimento prático com os alunos gerou dados que confirmam a aceitação da hipótese desse trabalho, portanto o método da Composição de Resultado promove melhoria no desempenho da aprendizagem dos estudantes em atividades práticas de programação. Pois há uma diferença considerável entre as médias de acerto dos grupos experimentais e de controle. O teste demonstrou a diferença é estatisticamente significativamente, estimado em um desempenho de 17% acertos a mais no grupo experimental.

Considerando os objetivos específicos apresentados no Capítulo 1, temos os diversos itens que os atendem, como o objetivo de **Identificar funcionalidade e características para o método e ferramentas de software** foi estabelecido no Capítulo 2 - Referencial Teórico. O objetivo **Propor um método de resolução de problemas de programação** foi atendido pelo exposto no Capítulo 3 - O Método: Composição de Resultado.

O objetivo **Desenvolver ferramentas de suporte ao método** é apresentado no Capítulo 4 - Ambiente de Aprendizagem, e no ambiente foram implementados os recursos que atendem ao objetivo **Disponibilizar recursos de suporte para as atividades do professor**. Além do objetivo **Fornecer mecanismos de correção automatizada para os estudantes** que foi implementado por meio dos testes baseados em entradas e saídas.

O objetivo **Apresentar orientações para condução da resolução** é atendido pelo feedback que a ferramenta gera, especificamente nos que indicam em qual passo o estudante está, e que medidas deve tomar para resolver o problema. O objetivo **Propor indicador para avaliar a resposta do estudante** por meio da avaliação Ciclomatica.

O objetivo **Demonstração da melhoria de desemprenho por meio de experimento** foi concebido no item Subseção 5.1.2 - Análise de Desempenho que mostrou bons resultados e validou a proposta desse estudo.

O objetivo **Levantar informações a respeito da satisfação de alunos e apreciação de professores** foi alcançado por meio da execução dos questionários: Apêndice C - Questionário de Satisfação para Estudante e Apêndice D - Questionário de Apreciação para Professores

Os questionários respondidos pelos professores demonstraram uma apreciação muito positiva, principalmente, nos quesitos de importância dos feedbacks, possibilidades de melhoria e interesse de uso em suas realidades de trabalho. Entretanto, indicou um ponto crucial de melhoria no método, no que se refere a apresentação de como ele funciona para favorecer o entendimento dos estudantes.

Considerando a necessidade de melhoria no entendimento dos passos do método, optamos por abstrair a explicação em separado da Composição de Resultado. Assim, dando prioridade a orientar os estudantes a fazer uso das ferramentas que se baseiam no método, deste modo fazendo uso indireto desse mecanismo de resolução de problemas.

No questionário dos estudantes foi revelado uma alto grau de importância dado a correção automatizada, considerando os feedbacks de erros e acertos para orientar a construção das respostas.

A método de Composição de Resultado proposto mostra-se promissor, visto que ele foi definido por meio de observações de como os estudantes resolviam os problemas, além de ser incrementada com experiências de desenvolvimento de software. E como meio de potencializa-lo, foi agregada a correção automatizada que faz os testes de modo incremental, e ainda oferece sugestão de correção baseada no teste em que falhou.

O ferramental técnico desenvolvido apresenta inúmeras vantagens, para o professor provê o monitoramento em tempo real das atividades que estão sendo realizadas, desempenho dos estudantes e indicadores para intervenções. Como a revisão de temas relacionados aos erros mais frequentes. Para o estudante, proporciona uma experiência rica, em um ambiente de programação tanto educacional quanto profissional, utilizando o método de Composição de resultado para auxiliar a construção da resposta de um problemas, e tendo feedbacks no decorrer das atividades.

6.1 Trabalhos Futuros

Como trabalhos futuros temos diversas oportunidades de melhoria e continuação deste trabalho. Entretanto se faz necessário um trabalho de finalização no sentido de correção de pequenos erros, para que os produtos desse estudos sejam usados efetivamente em turmas de ensino e treino de programação.

O método e as ferramentas podem ser usada tanto para treino quanto para avaliação de aprendizagem (provas práticas e competições de programação). É necessário imple-

mentar um robusto mecanismo de criptografia na troca de mensagens de comunicação, para diminuir possíveis tentativas de fraudes e intervenções maliciosas.

Uma interessante abordagem de análise de similaridade de código pode ser agregada e prover melhorias, para indicar o quanto a resposta de um estudante está próxima de demais respostas corretas. Ou indicar padrões de erros nas respostas e assim gerar sugestões de contorno e intervenções de correção.

Atualmente a ferramenta pode indicar que uma resposta está parcialmente correta, ou seja, passou em um determinada quantidade de testes. Assim o educador pode atribuir uma nota intermediária em relação a questão totalmente correta. Contudo se faz necessário implementar um mecanismo de atribuição de notas para as respostas dos estudantes. Deste modo podendo considerar a quantidade de testes com sucesso, quantidade de tentativas e métricas relacionadas a qualidade do código para mensurar uma nota adequada para cada resposta.

Como forma de avaliar os feedbacks fornecidos pela ferramenta, deve-se desenvolver um mecanismo em que os estudantes possam indicar o quanto o feedback ajudou a resolver o problema, e assim indicando oportunidades de melhorias para intervenção dos educadores.

6.2 Artigos Publicados

Como resultados deste trabalho, temos a publicação de um artigo em revista de educação. Sendo ela:

SOUSA, Rafael Gomes; FAVERO, Eloi Luiz. Resolução de problemas de programação com o método de composição de resultado. **RENOTE**, v. 13, n. 1. Ocorrida no segundo semestre de 2015.

Referências

AHARONI, D. Cogito, Ergo sum! cognitive processes of students dealing with data structures. *ACM SIGCSE Bulletin*, v. 32, n. 1, p. 26–30, mar 2000. Citado 2 vezes nas páginas 2 e 3.

BECK, K. *Test-driven development: by example*. 5. ed. Boston: Addison-Wesley Professional, 2003. 220 p. Citado na página 20.

BENNEDSEN, J.; CASPERSEN, M. E. Programming in context: a model-first approach to CS1. *SIGCSE 2004*, v. 36, p. 477–481, 2004. Citado 2 vezes nas páginas 19 e 20.

BORGES, M. Avaliação de uma metodologia alternativa para a aprendizagem de programação. *VIII Workshop de Educação em Computação–WEI*, 2000. Citado na página 3.

BOUD, D.; FELETTI, G. *The challenge of problem-based learning*. [S.l.]: Psychology Press, 1997. Citado na página 10.

BOWER, M. A taxonomy of task types in computing. *SIGCSE Bulletin*, v. 40, n. 3, p. 281–285, 2008. Citado na página 3.

CHI, M. T. H.; GLASER, R.; FARR, M. J. *The nature of expertise*. [S.l.]: Psychology Press, 2014. Citado na página 7.

CORMEN, T. H. et al. Algoritmos: teoria e pratica. *Editora Campus*, v. 3, p. 944, 2012. Citado na página 19.

DEEK, F. P. *An integrated environment for problem-solving and program development*. [S.l.: s.n.], 1997. Citado 3 vezes nas páginas x, 11 e 12.

DESAI, C.; JANZEN, D.; SAVAGE, K. A survey of evidence for test-driven development in academia. *ACM SIGCSE Bulletin*, v. 40, n. 2, p. 97, 2008. ISSN 00978418. Citado na página 11.

DOUCE, C.; LIVINGSTONE, D.; ORWELL, J. Automatic test-based assessment of programming: a review. *ACM Journal of Educational Resources in Computing*, v. 5, n. 3, p. 1–13, 2005. Citado na página 3.

EASTMAN, E. G. Fact-based problem identification precedes problem solving. *Journal of Computing Sciences in Colleges*, Consortium for Computing Sciences in Colleges, v. 19, n. 2, p. 18–29, 2003. Citado na página 13.

ECKERDAL, A. *Novice Programming Students' Learning of Concepts and Practise*. 194 p. Tese (PhD) — Uppsala UniversityUppsala University, 2009. Citado na página 3.

EDWARDS, S. H. Using software testing to move students from trial-and-error to reflection-in-action. *ACM SIGCSE Bulletin*, v. 36, n. 1, p. 26, 2004. Citado na página 2.

GOMES, A.; MENDES, A. J. N. Learning to program-difficulties and solutions. *International Conference on Engineering Education*, 2007. Citado na página 2.

GROSS, P.; POWERS, K. Evaluating assessments of novice programming environments. *Proceedings of the 2005 international workshop on Computing education research - ICER '05*, p. 99–110, 2005. Citado na página 3.

HICKEY, T. J. Scheme-based web programming as a basis for a CS0 curriculum. *ACM SIGCSE Bulletin*, v. 36, n. 1, p. 353, mar 2004. Citado na página 2.

HONG, N. S. The relationship between well-structured and ill-structured problem solving in multimedia simulation. *Knowledge Creation Diffusion Utilization*, n. August, p. 101, 1998. ISSN 1308-0911. Citado na página 8.

IHANTOLA, P. et al. Review of recent systems for automatic assessment of programming assignments. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research Koli Calling 10*, p. 86–93, 2010. Citado na página 3.

JONASSEN, D. Toward a design theory of problem solving. *Educational Technology Research and Development*, v. 48, n. 4, p. 63–85, 2000. ISSN 1042-1629. Citado na página 8.

KNUTH, D. E. *The art of computer programming*. [S.l.]: Addison-Wesley Publishing Company, 1968. Citado na página 7.

LAHTINEN, E.; ALA-MUTKA, K.; JARVINEN, H.-M. A study of the difficulties of novice programmers. *ACM SIGCSE Bulletin*, v. 37, n. 3, p. 14–18, 2005. Citado 2 vezes nas páginas 1 e 3.

LIKERT, R. *A technique for the measurement of attitudes*. 1932. 55 p. Disponível em: <http://psycnet.apa.org/psycinfo/1933-01885-001>. Citado na página 58.

LINDER, S. P. et al. Facilitating active learning with inexpensive mobile robots. *Journal of Computing Sciences in Colleges*, v. 16, n. 4, p. 21–33, 2001. Citado 2 vezes nas páginas 8 e 9.

MANN, H. B.; WHITNEY, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics*, JSTOR, p. 50–60, 1947. Citado na página 57.

MAZARIO, I. La resolución de problemas: un reto para la educación matemática contemporánea. *Universidad de La Habana*, p. 1–22, 2002. Citado na página 7.

MCCABE, T. J. A Complexity Measure. *IEEE Transactions on Software Engineering*, SE-2, n. 4, p. 308–320, 1976. ISSN 0098-5589. Citado na página 52.

MENDONÇA, A. P. et al. Dealing with Requirements Specification: A Case Study with Novice Programming Students. *IEEE Technology and Engineering Education (ITEE)*, v. 5, n. 1, p. 1–8, 2010. Citado na página 7.

MOTA, M. P.; PEREIRA, L. W. K.; FAVERO, E. L. Javatool: uma ferramenta para ensino de programação. *XXVIII Congresso da Sociedade Brasileira de ComputaComputação*, p. 127–136, 2008. Citado 3 vezes nas páginas x, 13 e 14.

MYERS, B. A. Taxonomies of visual programming and program visualization. *Journal of Visual Languages and Computing*, v. 1, n. 1, p. 97–123, 1990. ISSN 1045926X. Citado na página 9.

MYERS, G. J.; THOMAS, T. M.; SANDLER, C. *The Art of Software Testing*. [S.l.: s.n.], 2004. v. 1. 255 p. ISSN 0960-0833. ISBN 0471469122. Citado na página 11.

NETO, E. L. S. iProgram: uma ferramenta de apoio à avaliação de exercícios de programação. Universidade Federal de Pernambuco, 2015. Citado 3 vezes nas páginas x, 16 e 17.

NETO, V. L. et al. POPT: A Problem-Oriented Programming and Testing approach for novice students. In: *Proceedings - International Conference on Software Engineering*. [S.l.: s.n.], 2013. p. 1099–1108. ISBN 9781467330763. ISSN 02705257. Citado na página 18.

NEWMAN, M. J. Problem Based Learning: an introduction and overview of the key features of the approach. *Journal of veterinary medical education*, v. 32, n. 1, p. 12–20, 2005. ISSN 0748-321X. Citado na página 10.

PAPERT, S. *Mindstorms: Children, computers and powerful ideas*. [S.l.: s.n.], 1983. v. 1. 87 p. ISSN 0732118X. ISBN 0465046274. Citado na página 9.

PIETERSE, V. Automated Assessment of Programming Assignments. In: *3rd Computer Science Education Research Conference on Computer Science Education Research*. [S.l.: s.n.], 2013. v. 3, n. April, p. 45–56. Citado na página 3.

POLYA, G. *How to solve it: A new aspect of mathematical method*. [S.l.]: Princeton university press, 1978. Citado 2 vezes nas páginas 11 e 13.

POZO, J. I. A solucao de problemas: aprender a resolver, resolver para aprender. *Porto Alegre: Artmed*, 1998. Citado na página 7.

PRESSMAN, R. S. *Engenharia de Software: uma abordagem profissional*. [S.l.: s.n.], 2011. ISSN 85860457. ISBN 8586804576. Citado na página 11.

PRICE, B.; BAECKER, R.; SMALL, I. *An introduction to software visualization*. [S.l.]: Mit Press, 1998. 3–27 p. Citado na página 9.

PROULX, V. K. Programming patterns and design patterns in the introductory computer science course. *ACM SIGCSE Bulletin*, v. 32, n. 1, p. 80–84, 2000. Citado na página 1.

QUEIRÓS, R. A. P.; LEAL, J. P. PETCHA: a programming exercises teaching assistant. In: ACM. *Proceedings of the 17th ACM annual conference on Innovation and technology in computer science education*. [S.l.], 2012. p. 192–197. Citado 4 vezes nas páginas x, 15, 16 e 53.

RESNICK, M. et al. Programmable Bricks: Toys to think with. *IBM Systems Journal*, v. 35, n. Figure 1, p. 443–452, 1996. ISSN 0018-8670. Citado na página 9.

ROCHA, P. S. et al. Ensino e Aprendizagem de Programação: Análise da Aplicação de Proposta Metodológica Baseada no Sistema Personalizado de Ensino. *RENOTE - Revista Novas Tecnologias na Educação*, v. 8, n. 3, p. 1–11, 2010. ISSN 1679-1916. Disponível em: <http://seer.ufrgs.br/renote/article/view/18061>. Citado na página 54.

ROYSTON, J. P. Algorithm AS 181: the W test for normality. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, JSTOR, v. 31, n. 2, p. 176–180, 1982. Citado na página 57.

SAKAI, M. H.; LIMA, G. Z. PBL: uma visão geral do método. *Olho Mágico, Londrina*, v. 2, n. 5/6, p. 24–30, 1996. Citado na página 10.

SALTER, D.; DANTAS, R. *Netbeans IDE 8 Cookbook*. [S.l.]: Packt Publishing Ltd, 2014. Citado na página 35.

SANTOS, R. P. dos; COSTA, H. A. X. Análise de Metodologias e Ambientes de Ensino para Algoritmos , Estruturas de Dados e Programação aos iniciantes em Computação e Informática. *Infocomp Journal of Computer Science*, v. 5, p. 41–50, 2006. Citado na página 3.

SCAICO, P. D. et al. Programação no Ensino Médio: Uma Abordagem de Ensino Orientado ao Design com Scratch. *Anais do Congresso Brasileiro de Informática na Educação. XVIII Workshop de Informática na Escola*, p. 1–10, 2012. Citado na página 1.

SINGH, R.; GULWANI, S.; SOLAR-LEZAMA, A. Automated feedback generation for introductory programming assignments. *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation - PLDI '13*, p. 15, 2013. Citado na página 3.

SUDOL, L. A. Deepening Students' Understanding of Algorithms: Effects of Problem Context and Feedback Regarding Algorithmic Abstraction. *Thesis*, n. June, 2011. Citado na página 2.

TRUONG, N.; ROE, P.; BANCROFT, P. Automated Feedback for "Fill in the Gap"Programming Exercises. In: *Conferences in Research and Practice in Information Technology Series*. [S.l.: s.n.], 2005. v. 42, p. 117–126. Citado na página 3.

WOLZ, U. Teaching design and project management with lego RCX robots. *ACM SIGCSE Bulletin*, v. 33, n. 1, p. 95–99, 2001. ISSN 00978418. Citado 2 vezes nas páginas 9 e 52.

YEHEZKEL, C. A taxonomy of computer architecture visualizations. In: *Proceedings of the 7th annual conference on Innovation and technology in computer science education - ITiCSE'02*. [s.n.], 2002. p. 101. ISBN 1581134991. ISSN 0097-8418. Disponível em: <http://dl.acm.org/citation.cfm?id=544414.544447>. Citado na página 9.

Apêndice A

Ficha de Avaliação do Problema

Segue a ficha que foi usada para auxiliar a análise do problema.

Ficha de Avaliação do Problema

Problema:_____

Nome do Programa:_____

Item 1: Destaque as variáveis de **Entrada**:

Descrição	Tipo (int, float, char, String, boolean, etc...)	Nome da Variável	Valor Inicial

Item 2: Destaque os Valores de **Saída**:

Descrição	Ex: Tipo ou Formatação de saída

Item 3: Identifique o **cálculo** que o programa deve realizar (Baseado nos elementos de entrada/saída):

Item 4: O programa apresenta alguma **pergunta** que deve ser respondida (Decisão, Critério, Restrição)?
Indique(circule) qual estrutura de **SELEÇÃO** deve ser usada?

Seleção Simples	**Seleção Dupla/Composta**	**Seleção Múltipla**

`if (numero > 7) {`
`  System.out.println("maior que 7");`
`}`
	`if (numero > 7) {`
	`  System.out.println("maior que 7");`
	`} else {`
	`  System.out.println("<=a 7");`
	`}`
		`switch (opcao) {`
		`  case 1:`
		`  System.out.println("Opcao 1");`
		`  break;`
		`  default:`
		`  System.out.println("Opcao invalida");`
		`}`

Uma pergunta com uma caminho possível de execução.	Uma pergunta com dois caminhos de execução. **Nota**: Se houver mais de uma pergunta dependente, será necessário encadear if e else.	Seleciona um grupo de ações dependendo da avalição de valor de opção, oferecendo múltiplos caminhos de execução. **Nota**: Não consegue trabalhar com faixas de valores.

Item 5: O programa apresenta alguma **instrução** que deva ser **executada mais** de **uma vez**?
Indique(circule) qual estrutura de **REPETIÇÃO** deve ser usada?

Repetição: while	**Repetição: do..while**	**Repetição: for**

`//Controlada por Contador`
`int i = 1;//Criacao var. controle`
`while (i <= 10) {//teste de continuidade`
`  System.out.println(i);`
`  i++;//incremnteo do controle`
`}`
`// Controlada por Flag`
`while (numero != -1) {`
`  System.out.println(i);`
`  numero = entrada.nextInt();`
`}`
	`int i = 1;`
	`do {`
	`  System.out.println(i);`
	`  i++;`
	`} while (i <= 10);`
		`for (int i = 0; i <= 10; i++) {`
		`  System.out.println(i);`
		`}`

Resolve qualquer problema de repetição, entretanto é mais indicado para repetição controlada por Flag	Semelhante ao laço while, mas executa as instruções pelo menos uma vez e possui o teste lógico no final.	Laço especialista em repetição controlada por Contador

Apêndice B

Ficha de Resumo do Método

Segue a ficha que foi usada para auxiliar a análise do problema relacionada com o método de resolução.

Ficha de Resumo do Método de Composição do Resultado

Passo 1: Dividir o problema

Após destacar os **elementos de saída** (Item 2 da Ficha de Avaliação do Problema) os subproblemas podem ser destacados a partir do que é esperado como saída.

Informe quais seriam os possíveis subproblemas, ou seja, quais partes formarão a saída:

Passo 2: Conceber a base do programa

Crie um programa com as informações básicas destacadas na Ficha de Avaliação do Problema, com o mínimo de código possível, ou seja, com a **leitura** das **variáveis de entrada** e **escrita** da **saída** no formato que o problema pede. Sempre fazendo referência a entrada e saída vazias, por exemplo lendo as variáveis de entrada iguais a zero e escrevendo o resultado com zeros.

Passo 3: Resolução dos subproblemas

Passo 3.1: Construir a resposta para um subproblema

Após identificar os possíveis subproblemas agora você deve **escolher** o mais **simples** para **implementar**, ou seja, ira implementar a porção de código favoreça a resolução de parte da saída.

Passo 3.2: Realizar um ou mais testes focados nesse subproblema

Assim que uma parte do problema é implementada podemos testar com **entradas** que **favoreçam** o **teste** dele, visto que inicialmente trabalhamos com entrada e saída vazias, se implementamos uma parte, essa parte será o foco do teste pois as demais ainda estarão trabalhando com a entrada e saída vazias.

Passo 4: Integrar as respostas dos subproblemas

Talvez seja necessário fazer alguma **alteração** para que as **partes** do problemas possam ser **usadas** em **conjunto**, além disso realize **testes** considerando todas as **possibilidades** de **entrada**, visando averiguar se as partes juntas continuam produzindo a resposta correta.

Nota 01: Se não conseguiu identificar o cálculo que o programa deve realizar, considere fazer uma pesquisa sobre a temática do problema.

Nota 02: Se identificar um estrutura de seleção (pergunta) verifique quantas são e se elas dependem uma da outra, pois isso pode orientar a escolha da estrutura mais adequada.

Nota 03: Se identificar uma estrutura de repetição foque no que terá que ser repetido, pois essas instruções formarão o corpo do laço. Dê preferência em tentar realizar a repetição com o menor número de iterações possíveis, desta forma será mais simples de testar.

Apêndice C

Questionário de Satisfação para Estudante

Questionário de Satisfação para Estudante

Esta coleta de informações destina-se a produzir dados para análise e melhoria das ferramentas e metodologia, aplicadas ao ensino e prática de programação de computadores.

Sigilo

Todas as informações coletadas serão mantidas em sigilo, deste modo só serão utilizadas para compor estatísticas.

Sexo: ( ) M ( ) F
Idade: ( ) Até 20
( ) 21 de 30
( ) 31 de 40
( ) 41 de 50
( ) Mais de 50

1) Como avalia o seu **conhecimento** teórico sobre os **conceitos de programação**?

( ) Extremamente Satisfatório

( ) Muito Satisfatório

( ) Moderadamente Satisfatório

( ) Pouco Satisfatório

( ) Nada Satisfatório

2) Como avalia o seu grau de **entendimento** do enunciado dos problemas de programação e a aplicação das estruturas de programação na resolução dele?

( ) Extremamente Satisfatório

( ) Muito Satisfatório

( ) Moderadamente Satisfatório

( ) Pouco Satisfatório

( ) Nada Satisfatório

3) Como avalia a sua **dificuldade** para o **entendimento** do método da **Composição de Resultado** para resolver problemas de programação?

( ) Extrema dificuldade

( ) Muita dificuldade

( ) Moderada dificuldade

( ) Pouca dificuldade

( ) Nenhuma dificuldade

4) Como avalia a **dificuldade** de **instalação/uso** das **ferramentas de software** apresentadas nesse curso (LabProg Web e Plug-in p/ Netbeans)?

( ) Extrema dificuldade

( ) Muita dificuldade

( ) Moderada dificuldade

( ) Pouca dificuldade

( ) Nenhuma dificuldade

5) Como avalia o grau de **importância** dos **feedbacks** que foram apresentados para a **resolução** dos **problemas** de programação, considerando **iniciar** e **progredir** a construção das respostas?

( ) Extrema importância

( ) Muito importante

( ) Moderada importância

( ) Pouca importância

( ) Nenhuma importância

6) Como avalia o grau de **importância** do **indicativo** que demonstra se o programa está **correto** ou quantos **testes restam** para que a solução seja considerada **completa**?

( ) Extrema importância

( ) Muito importante

( ) Moderada importância

( ) Pouca importância

( ) Nenhuma importância

7) Considerando o **método** e as **ferramentas**, como avalia o seu grau de **melhoria** nas **atividades práticas** de **programação**?

( ) Extrema melhoria

( ) Muita melhoria

( ) Moderada melhoria

( ) Pouca melhoria

( ) Nenhuma melhoria

8) Considerando o **método** e as **ferramentas**, você teria **interesse** em continuar fazendo **uso** para **treinar** suas habilidades de **resolução** de problemas?

( ) Extremo interesse

( ) Muito interesse

( ) Moderado interesse

( ) Pouco interesse

( ) Nenhum interesse

Apêndice D

Questionário de Apreciação para Professores

Questionário de Satisfação para Professores

Esta coleta de informações destina-se a produzir dados para análise e melhoria das ferramentas e metodologia, aplicadas ao ensino e prática de programação de computadores.

Sigilo

Todas as informações coletadas serão mantidas em sigilo, deste modo só serão utilizadas para compor estatísticas.

Sexo: ( ) M ( ) F
Idade: ( ) Até 20
( ) 21 de 30
( ) 31 de 40
( ) 41 de 50
( ) Mais de 50

Tempo de ensino em programação: ___ Anos

1) Como avalia o grau de dificuldade para o **entendimento** do método da **Composição de Resultado** para resolver problemas de programação?

( ) Extrema dificuldade

( ) Muita dificuldade

( ) Moderada dificuldade

( ) Pouca dificuldade

( ) Nenhuma dificuldade

2) Como avalia a **dificuldade** de **instalação/uso** das **ferramentas de software** apresentadas nesse (LabProg Web e Plug-in p/ Netbeans)?

( ) Extrema dificuldade

( ) Muita dificuldade

( ) Moderada dificuldade

( ) Pouca dificuldade

( ) Nenhuma dificuldade

3) Como avalia o grau de **importância** dos **feedbacks** que foram apresentados para a **resolução** dos **problemas** de programação, considerando **iniciar** e **progredir** a construção das respostas?

( ) Extrema importância

( ) Muita importância

( ) Moderada importância

( ) Pouca importância

( ) Nenhuma importância

4) Como avalia o grau de **importância** do **indicativo** que demonstra se o programa está **correto** ou quantos **testes restam** para que a solução seja considerada **completa**?

( ) Extrema importância

( ) Muita importância

( ) Moderada importância

( ) Pouca importância

( ) Nenhuma importância

5) Considerando o **método** e as **ferramentas**, como avalia o grau de melhoria pode ser **ofertado** em atividades práticas de programação?

( ) Extrema melhoria

( ) Muita melhoria

( ) Moderada melhoria

( ) Pouca melhoria

( ) Nenhuma melhoria

6) Considerando o **método** e as **ferramentas**, qual grau de **interesse** teria em fazer **uso** em um curso de programação?

( ) Extremo interesse

( ) Muito interesse

( ) Moderada interesse

( ) Pouco interesse

( ) Nenhum interesse

Complemento _____

Apêndice E

Problemas das listas de Programação

E.1 Lista1

Soma

Faça um programa que leia dois números inteiros digitados pelo teclado e imprima a soma deles.

Entrada

A entrada consiste de dois números inteiros X e Y.

Saída

A saída do seu programa deve ser um inteiro S, representando a soma de X e Y.

Exemplos

Entrada 0 0	**Saída** 0
Entrada 2 0	**Saída** 2
Entrada 0 3	**Saída** 3
Entrada 2 3	**Saída** 5

Calibrando Pneus

Calibrar os pneus do carro deve ser uma tarefa cotidiana de todos os motoristas. Para isto, os postos de gasolina possuem uma bomba de ar. A maioria das bombas atuais são eletrônicas, permitindo que o motorista indique a pressão desejada num teclado. Ao ser ligada ao pneu, a bomba primeiro lê a pressão atual e calcula a diferença de pressão entre a desejada e a lida. Com esta diferença ela esvazia ou enche o pneu para chegar na pressão correta. Sua ajuda foi requisitada para desenvolver o programa da próxima bomba da SBC - Sistemas de Bombas Computadorizadas. Escreva um programa que, dada a pressão desejada digitada pelo motorista e a pressão do pneu lida pela bomba, indica a diferença entre a pressão desejada e a pressão lida.

Entrada

A primeira entrada contém um inteiro N que indica a pressão desejada pelo motorista. A segunda entrada contém um inteiro M que indica a pressão lida pela bomba.

Saída

Seu programa deve imprimir uma única linha, contendo a diferença entre a pressão desejada e a pressão lida.

Exemplos

Entrada 0 0	**Saída** 0
Entrada 6 0	**Saída** 6
Entrada 0 8	**Saída** -8
Entrada 29 10	**Saída** 19
Entrada 32 32	**Saída** 0

Fonte: Tarefa da OBI2010, Modalidade Programação Júnior, Fase 1

Média Aluno

Construa um programa que faça a média da final(aritmética) para um estudante sabendo-se que ele realiza apenas duas avaliações para uma determinada disciplina.

Entrada

A entrada ocorre com o usuário informando dois valores de notas referente as avaliações.

Saída

Seu programa deve imprimir um único valor informando o valor da média final do aluno.

Exemplos

Entrada 0 0	**Saída** 0,0
Entrada 10 0	**Saída** 5,0
Entrada 0 8	**Saída** 4,0
Entrada 9 9	**Saída** 9,0
Entrada 5,66 8,77	**Saída** 7,2

Metade maior que 20

Escreva um programa que leia um número e se ele for maior do que 20, então imprimir a metade do número.

Entrada

A única entrada contendo um número N.

Saída

Seu programa deve escrever uma única saída contendo o resultado solicitado.

Exemplos

Entrada 0	**Saída**
Entrada 10	**Saída**
Entrada 50	**Saída** 25
Entrada 25	**Saída** 12
Entrada 100	**Saída** 50

E.2 Lista2

Aprovado

Construa um programa que informe se um aluno foi aprovado, considerando que ele conseguiu uma média maior ou igual a 6(seis).

Entrada

A entrada consiste em três notas A, B e C que correspondem as avaliações que o aluno realizou durante o semestre.

Saída

A saída do seu programa deve ser apenas a informação se ele foi aprovado "Aprovado”.

Exemplos

Entrada 0 0 0	**Saída**
Entrada 10 0 0	**Saída**
Entrada 10 10 0	**Saída** Aprovado
Entrada 7 8 9	**Saída** Aprovado

Raiz do Negativo

Escreva um programa que leia um número e informe a sua raiz quadrada, entretanto considerando a inviabilidade de calcular raiz quadrada de números negativos, devemos converter esse para um número positivo e dessa forma obter a raiz de qualquer número.

Entrada

A única entrada contendo um número X.

Saída

Seu programa deve escrever uma única saída contendo a raiz quadrada de X.

Exemplos

Entrada 0	**Saída** 0,00
Entrada 1	**Saída** 1,00
Entrada -16	**Saída** 4,00
Entrada 20	**Saída** 4,47
Entrada -64	**Saída** 8,00

Triplo do maior

Escreva um programa que leia dois números e informe uma na saída o valor contendo o triplo do maior dos dois valores lidos da entrada.

Entrada

A entrada contém dois números inteiros distintos A e B, considera A e B sempre diferentes.

Saída

Seu programa deve escrever na saída resultado do cálculo solicitado.

Exemplos

Entrada 0 0	**Saída** 0
Entrada 3 0	**Saída** 9
Entrada 0 4	**Saída** 12
Entrada 8 9	**Saída** 27

Jogo do Par ou Ímpar

João e Maria jogaram Par ou Ímpar, e em todos os jogos João escolheu ímpar (e consequentemente Maria escolheu par). Assim em uma partida cada um deles levanta uma determinada quantidade de dedos no momento que ambos falam "já".

Entrada

A entrada é composta por duas variáveis indicando a quantidade de dedos informadas por João e por Maria.

Saída

Seu programa deve produzir uma saída informando zero se Maria ganhou e o valor um se foi João.

Exemplos

Entrada 0 0	**Saída** 0
Entrada 3 0	**Saída** 1
Entrada 0 2	**Saída** 0
Entrada 3 4	**Saída** 1

E.3 Lista3

Aprovado ou Reprovado

No colégio Barias Frito (BF) um aluno é aprovado por média se ele obtiver uma média final maior ou igual a 7, caso o aluno tenha uma média menor que 7, mas maior ou igual a 5 ele está de recuperação, caso ele tenha uma média menor que 5 o aluno está reprovado.

A média é calculada com a nota das duas provas aplicadas no semestre e corresponde simplesmente a média aritmética das duas notas.

Baseado nas duas notas do aluno, indique o resultado final do aluno: "Aprovado", "Reprovado" ou "Recuperacao".

Entrada

A entrada consiste de apenas uma linha com as notas A e B, que correspondem as duas notas que o aluno conquistou esse semestre.

Saída

A saída do seu programa deve ser apenas uma linha. Caso o aluno tenha sido aprovado informe "Aprovado", caso o aluno tenha sido reprovado informe "Reprovado" e caso ele esteja de recuperação informe "Recuperacao".

Exemplos

Entrada 0 0	**Saída** Reprovado
Entrada 0 8	**Saída** Reprovado
Entrada 10 0	**Saída** Recuperacao
Entrada 10 10	**Saída** Aprovado

IMC

O índice de massa corporal, mais conhecido pela sigla IMC, é um índice adotado pela OMS (Organização Mundial de Saúde), que é usado para o diagnóstico do sobrepeso. O IMC pode ser facilmente calculado a partir de dois simples dados: peso e altura. E o cálculo é realizado dividindo o peso pela altura elevada ao quadrado.

Essa informação é essencial para indicar possíveis riscos de saúde, e temos a identificação de um grupo de risco para pessoas que apresentam IMC com valor depois de 25, com pessoas que estão acima do peso. Construa um programa que informe se um indivíduo está com sobrepeso baseando-se nas informações de peso e altura.

Entrada

A entrada contém duas variáveis, o número P que corresponde ao peso e um numero A com a informação da altura.

Saída

Seu programa deve escrever uma única saída contendo o "Sobrepeso" para os indivíduos que se encontram no grupo de risco.

Exemplos

Entrada 0 1	**Saída**
Entrada 59 1,65	**Saída**
Entrada 100 1,92	**Saída** Sobrepeso
Entrada 70 1,67	**Saída** Sobrepeso

Tomadas

A Olimpíada Internacional de Informática (IOI, no original em inglês) é a mais prestigiada competição de programação para alunos de ensino médio; seus aproximadamente 300 competidores se reúnem em um país diferente todo ano para os dois dias de prova da competição. Naturalmente, os competidores usam o tempo livre para acessar a Internet, programar e jogar em seus notebooks, mas eles se depararam com um problema: o saguão do hotel só tem uma tomada.

Felizmente, os quatro competidores da equipe brasileira da IOI trouxeram cada um uma régua de tomadas, permitindo assim ligar vários notebooks em uma tomada só; eles também podem ligar uma régua em outra para aumentar ainda mais o número de tomadas disponíveis. No entanto, como as réguas têm muitas tomadas, eles pediram para você escrever um programa que, dado o número de tomadas em cada régua, determina quantas tomadas podem ser disponibilizadas no saguão do hotel

Entrada

A entrada ocorre com o usuário informando dois valores de notas referente as avaliações. A entrada consiste de uma linha com quatro inteiros positivos T_1, T_2, T_3, T_4, indicando o número de tomadas de cada uma das quatro réguas. Nota: os valores de T estão entre 2 e 6.

Saída

Seu programa deve imprimir uma única linha contendo um único número inteiro, indicando o número máximo de notebooks que podem ser conectados num mesmo instante.

Exemplos

Entrada 2 2 2 2	**Saída** 5
Entrada 3 2 2 2	**Saída** 6
Entrada 2 2 2 4	**Saída** 7
Entrada 2 4 3 2	**Saída** 8
Entrada 2 2 4 5	**Saída** 10
Entrada 6 6 6 6	**Saída** 21

Tarefa da OBI2013, Modalidade Programação Júnior, Fase 1

Vice-líder

A OBI (Organização de Bocha Internacional) é responsável por organizar a competição mundial de bocha. Infelizmente esse esporte não é muito popular, e numa tentativa de aumentar a sua popularidade, ficou decidido que seriam chamados, para a Grande Final Mundial, o campeão e o vice-campeão de cada sede nacional, ao invés de apenas o primeiro lugar.

Tumbólia é um país pequeno que já havia realizado a sua competição nacional quando a nova regra foi instituída, e o comitê local não armazenou quem foi o segundo classificado. Felizmente eles armazenaram a pontuação de todos competidores - que foram apenas três, devido ao tamanho diminuto do país. Sabe-se também que as pontuações de todos jogadores foram diferentes, de forma que não ocorreu empate entre nenhum deles.

Resta agora descobrir quem foi o vice-campeão e para isso o comitê precisa de ajuda.

Entrada

A primeira e única linha da entrada consiste de três inteiros separados por espaços, *A*, *B* e *C*, as pontuações dos 3 competidores.

Saída

Imprima uma única linha na saída, contendo apenas um número inteiro, a pontuação do vice-campeão.

Exemplos

Entrada	**Saída**
1 2 3	2
Entrada	**Saída**
1 0 7	1
Entrada	**Saída**
10 5 9	9
Entrada	**Saída**
4 5 6	5

Fonte: Tarefa da OBI2012, Modalidade Programação Nível 1, Fase 1

E.4 Lista4

Controle de Qualidade

Maria trabalha em uma fábrica de automóveis, e o seu trabalho é qualificar os defeitos encontrados no final do processo de montagem dos carros. Ela conta quantos defeitos encontra na pintura e revestimento interne. Dependendo da quantidade da defeitos ela uma a seguinte tabela para qualificar os produtos:

Nota	Qualidade
A partir de 100	INAPROPRIADO
61 a 99	Classe C
1 a 60	Classe B
0	Classe A

Devemos escrever um programa que recebe uma nota no sistema numérico e determina o conceito correspondente.

Entrada

A entrada contém um valor inteiro indicando a quantidade de erros

Saída

Seu programa deve imprimir a qualidade correspondente ao número de erros.

Exemplos

Entrada	**Saída**
19	Classe B
Entrada	**Saída**
77	Classe C
Entrada	**Saída**
0	Classe A

Desafio da maior palavra

Aldo e Beatriz gostam de jogos de palavras. Um dos jogos que eles acham mais divertido é encontrar palavras com muitas letras. Por exemplo, eles definem que o tema do desafio é *nome de cidades*, e os dois tentam encontrar a maior palavra que seja nome de uma cidade. Vence o desafio quem encontrar maior palavra, ou seja a palavra com o maior número de letras.

Eles pediram que você escreva um programa que, dadas as duas palavras que Aldo e Beatriz encontraram, determine qual dos dois é o vencedor.

Entrada

A entrada consiste de apenas uma linha, contendo duas palavras A e B, respectivamente as palavras encontradas por Aldo e por Bia. Uma palavra é uma sequência de letras.

Saída

Seu programa deve imprimir uma única linha contendo um único caractere, que deve ser A se a maior palavra é a de Aldo, ou B se a maior palavra é a de Beatriz, ou o caractere `*` (asterisco) se as duas palavras têm o mesmo número de letras.

Nota

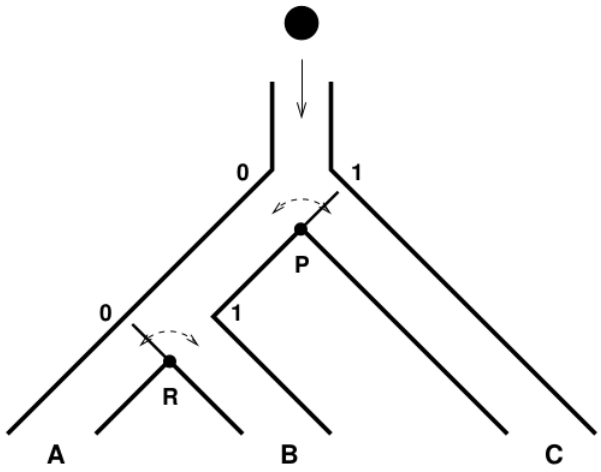
- Os caracteres são somente letras e não são acentuadas.

Exemplos

Entrada	**Saída**
`Pindamonhangaba SaoJoaoDaBoaVista`	`B`
Entrada	**Saída**
`TrepadeiraElefante` `CrisantemoVermelho`	`*`
Entrada	**Saída**
`Jabuticaba Uva`	`A`

Flíper

Flíper é um tipo de jogo onde uma bolinha de metal cai por um labirinto de caminhos até chegar na parte de baixo do labirinto. A quantidade de pontos que o jogador ganha depende do caminho que a bolinha seguir. O jogador pode controlar o percurso da bolinha mudando a posição de algumas portinhas do labirinto. Cada portinha pode estar na posição 0, que significa virada para a esquerda, ou na posição 1 que quer dizer virada para a direita. Considere o flíper da figura abaixo, que tem duas portinhas. A portinha P está na posição 1 e a portinha R, na posição 0. Desse jeito, a bolinha vai cair pelo caminho B.



Você deve escrever um programa que, dadas as posições das portinhas P e R, neste flíper da figura, diga por qual dos três caminhos, A, B ou C, a bolinha vai cair!

Entrada

A entrada é composta por apenas uma linha contendo dois números P e R, indicando as posições das duas portinhas do flíper da figura.

Saída

A saída do seu programa deve ser também apenas uma linha, contendo uma letra maiúscula que indica o caminho por onde a bolinha vai cair: 'A', 'B' ou 'C'.

Exemplos

Entrada	**Saída**
1 0	B

Entrada	**Saída**
0 0	C

A mais Pedida da Semana

Joe Moreno é um radialista famoso, e semanalmente recebe muitos pedidos de música em sua programação. Ele necessita saber qual foi a música mais pedida durante a semana, entretanto ele só tem em suas anotações a quantidade de vezes que cada música foi pedida. Assim, vamos ajudar desenvolvendo um programa para informar qual foi a música que mais tocou.

Entrada

A primeira variável de entrada é um inteiro **N**, que indica o número de músicas que foram tocadas na semana. E em seguida são apresentadas **N** inteiros informando quantas vezes cada música foi tocada.

Saída

O nosso programa deve produzir uma saída, contendo um único número, o maior número de vezes que uma música foi tocada.

Exemplos

Entrada	**Saída**
4 1000 4000 500 2500	4000

Entrada	**Saída**
6 0 1 6 6 2 3	6

E.5 Lista5

Divisores de N

Crie um programa que leia um valor e informe a sequência de divisores iniciando do número 1 (um) até o número lido.

Entrada

A entrada é composta por um número inteiro N.

Saída

A saída do programa deve conter apenas uma linha com os divisores de N separados por um espaço em branco.

Exemplos

Entrada 1	**Saída** 1
Entrada 2	**Saída** 1 2
Entrada 3	**Saída** 1 3
Entrada 6	**Saída** 1 2 3 6
Entrada 7	**Saída** 1 7
Entrada 20	**Saída** 1 2 4 5 10 20

Zerinho ou Um

Todos devem conhecer o jogo *Zerinho ou Um* (em algumas regiões também conhecido como *Dois ou Um*), utilizado para determinar um ganhador entre três ou mais jogadores. Para quem não conhece, o jogo funciona da seguinte maneira. Cada jogador escolhe um valor entre zero ou um; a um comando (geralmente um dos competidores anuncia em voz alta "Zerinho ou... Um!"), todos os participantes mostram o valor escolhido, utilizando uma das mãos: se o valor escolhido foi um, o competidor mostra o dedo indicador estendido; se o valor escolhido foi zero, mostra a mão com todos os dedos fechados. O ganhador é aquele que tiver escolhido um valor diferente de todos os outros; se não há um jogador com valor diferente de todos os outros (por exemplo todos os jogadores escolhem zero, ou um grupo de jogadores escolhe zero e outro grupo escolhe um), não há ganhador. Alice, Beto e Clara são grandes amigos e jogam Zerinho a toda hora: para determinar quem vai comprar a pipoca durante a sessão de cinema, quem vai entrar na piscina primeiro, etc. Jogam tanto que resolveram fazem um plugin no Facebook para jogar Zerinho. Como não sabem programar, dividiram as tarefas entre amigos que sabem, inclusive você.

Dados os três valores escolhidos por Alice, Beto e Clara, cada valor zero ou um, escreva um programa que determina se há um ganhador, e nesse caso determina quem é o ganhador.

Entrada

A entrada é composta por três inteiros A, B e C, indicando respectivamente os valores escolhidos por Alice, Beto e Clara.

Saída

Seu programa deve produzir uma única linha, contendo um único caractere. Se o vencedor é Alice o caractere deve ser 'A', se o vencedor é Beto o caractere deve ser 'B', se o vencedor é Clara o caractere deve ser 'C' e se não há vencedor o caractere deve ser '*' (asterisco).

Exemplos

Entrada 0 0 0	**Saída** *
Entrada 1 0 0	**Saída** A
Entrada 0 1 0	**Saída** B
Entrada 1 1 0	**Saída** C

Fonte: Tarefa da Maratona de Programação da SBC 2013, Fase Local

Desafio do Maior Número

Leonardo é um garoto muito criativo. Ele adora criar desafios para seus colegas da escola. Seu último desafio é o seguinte: diversos números são ditos em voz alta, quando o número 0 (zero) é dito então o desafio termina e seus colegas devem dizer imediatamente qual foi o maior número. Leonardo tem muita dificuldade de verificar se a resposta dada pelos colegas é correta ou não, pois a sequência de números costuma ser longa. Por este motivo, ele resolveu pedir sua ajuda.

Sua tarefa é escrever um programa que dada uma sequência de números inteiros positivos terminada por 0 (zero), imprime o maior número da sequência.

Entrada

A entrada é dada em uma única linha contendo uma sequência de números inteiros positivos. O último número da linha é 0 (zero).

Saída

Seu programa deve imprimir o maior número dentre os números da entrada.

Exemplos

Entrada 0	**Saída** 0
Entrada 2 0	**Saída** 2
Entrada 4 8 0	**Saída** 8
Entrada 10 30 20 5 0	**Saída** 30
Entrada 99 1000 55 1 2 9 0	**Saída** 1000

Fonte: OBI 2012, Fase 1, Nível Júnior.

Notas da prova

Rosy é uma talentosa professora do Ensino Médio que já ganhou muitos prêmios pela qualidade de sua aula. Seu reconhecimento foi tamanho que foi convidada a dar aulas em uma escola da Inglaterra. Mesmo falando bem inglês, Rosy ficou um pouco apreensiva com a responsabilidade, mas resolveu aceitar a proposta e encará-la como um bom desafio.

Tudo ocorreu bem para Rosy até o dia da prova. Acostumada a dar notas de 0 (zero) a 100 (cem), ela fez o mesmo na primeira prova dos alunos da Inglaterra. No entanto, os alunos acharam estranho, pois na Inglaterra o sistema de notas é diferente: as notas devem ser dadas como conceitos de A a E. O conceito A é o mais alto, enquanto o conceito E é o mais baixo. Conforme a tabela que segue:

Nota	Conceito
86 a 100	A
61 a 85	B
36 a 60	C
1 a 35	D
0	E

O problema é que Rosy já deu as notas no sistema numérico, e terá que converter as notas para o sistema de letras. Porém, Rosy precisa preparar as próximas aulas (para manter a qualidade que a tornou reconhecida), e não tem tempo suficiente para fazer a conversão das notas manualmente. Então você deve escrever um programa que recebe uma nota no sistema numérico e determina o conceito correspondente.

Entrada

A entrada contém uma única linha com um número inteiro N representando uma nota de prova no sistema numérico.

Saída

Seu programa deve imprimir uma única linha, contendo uma letra (A, B, C, D, ou E em maiúsculas) representando o conceito correspondente à nota dada na entrada.

Exemplos

Entrada 0	**Saída** E
Entrada 12	**Saída** D
Entrada 50	**Saída** C
Entrada 88	**Saída** A

E.6 Lista6

Conta de água

A empresa local de abastecimento de água, a Saneamento Básico da Cidade (SBC), está promovendo uma campanha para incentivar mais ainda a economia de água, a SBC alterou os preços de seu fornecimento de forma que, proporcionalmente, aqueles clientes que consumirem menos água paguem menos pelo metro cúbico. Todo cliente paga mensalmente uma assinatura de R$ 7, que inclui uma franquia de 10 m^3 de água. Isto é, para qualquer consumo entre 0 e 10 m^3, o consumidor paga a mesma quantia de R$ 7 reais (note que o valor da assinatura deve ser pago mesmo que o consumidor não tenha consumido água). Acima de 10 m^3, cada metro cúbico subsequente tem um valor diferente, dependendo da faixa de consumo. A SBC cobra apenas por quantidades inteiras de metros cúbicos consumidos. A tabela abaixo especifica o preço por metro cúbico para cada faixa de consumo:

Faixa de consumo (m^3)	**Preço (por m^3)**
até 10	incluído na franquia
11 a 30	R$ 1
31 a 100	R$ 2
101 em diante	R$ 5

Assim, por exemplo, se o consumo foi de 120 m^3, o valor da conta é:

- 7 reais da assinatura básica;
- 20 reais pelo consumo no intervalo 11 a 30 m^3;
- 140 reais pelo consumo no intervalo 31 a 100 m^3;
- 100 reais pelo consumo no intervalo 101 a 120 m^3.

Logo o valor total da conta de água é R$ 267.

Escreva um programa que, dado o consumo de uma residência em m^3, calcula o valor da conta de água daquela residência.

Entrada

A entrada contém um único *inteiro* N que indica o consumo de água em m^3.

Saída

Seu programa deve escrever uma única linha, contendo o valor da conta de água da residência.

Entrada 0	**Saída** 7
Entrada 8	**Saída** 7
Entrada 14	**Saída** 11
Entrada 42	**Saída** 51

Audiência

Joana criou um canal no YouTube, que tem feito muito sucesso! Desde o primeiro dia, diariamente Joana anota o número de visualizações de vídeos do seu canal, no que ela chamou de Lista de Audiência.

Joana quer saber se sua Lista de Audiência é estritamente crescente (ou seja, se cada elemento na lista é maior do que o elemento imediatamente anterior). Você pode ajudá-la?

Entrada

A entrada consiste de duas linhas. A primeira linha contém um número inteiro N que indica o comprimento da Lista de Audiência. A segunda linha descreve os elementos da Lista de Audiência. Ela contém N inteiros X_i, indicando o número de visualizações em dias consecutivos.

Saída

Seu programa deve escrever uma única linha na saída contendo apenas um caractere: "S" se a Lista de Audiência é crescente, ou "N" caso contrário.

Exemplos

Entrada	**Saída**
1 10	S
Entrada	**Saída**
2 10 11	S
Entrada	**Saída**
3 1 3 2	N
Entrada	**Saída**
4 1 1000 1000 2000	N
Entrada	**Saída**
3 3 2 1	N
Entrada	**Saída**
5 10 30 40 41 80	S

Saldo do Vovô

Vovô João tem uma banca de jornais; ele tem muitos clientes, e diariamente recebe muito dinheiro, mas também faz muitos pagamentos para manter o seu estoque de jornais e revistas. Todo dia ele vai ao banco realizar um depósito ou uma retirada de dinheiro. Em alguns dias, o saldo de sua conta no banco fica negativo, mas Vovô João tem um acordo com o banco que garante que ele somente é cobrado se o saldo for menor do que um valor pré-estabelecido.

Entrada

A primeira linha da entrada contém dois números inteiros N e S que indicam respectivamente o número de dias do período de interesse e o saldo da conta no início do período. Cada uma das N linhas seguintes contém um número inteiro indicando a movimentação de um dia (valor positivo no caso de depósito, valor negativo no caso de retirada). A movimentação é dada para um período de N dias consecutivos: a primeira das N linhas corresponde ao primeiro dia do período de interesse, a segunda linha corresponde ao segundo dia, e assim por diante.

Saída

Seu programa deve imprimir uma única linha, contendo um único número inteiro, o menor valor de saldo da conta no período dado.

Exemplos

Entrada 0 0	**Saída** 0
Entrada 2 0 100 -20	**Saída** 80
Entrada 3 200 -80 50 -150	**Saída** 20
Entrada 3 1000 100 -800 50	**Saída** 300
Entrada 6 -200 -100 1000 -1000 100 -50 1000	**Saída** -300

Fonte: OBI 2013, Fase 1, Nível Júnior.

Média Turma 10 Alunos

Faça um programa que leia 10 valores de notas de alunos e informe a média aritmética da turma, ou seja a soma das notas dividido pela quantidade de alunos.

Entrada

A entrada consiste de uma lista com 10 números que representam notas dos estudantes.

Saída

A saída do seu programa deve ser um valor que corresponde à média da turma.

Exemplos

Entrada 0 0 0 0 0 0 0 0 0 0	**Saída** 0,0
Entrada 10 0 0 0 0 0 0 0 0 0	**Saída** 1,0
Entrada 8 5 5 6 6 6 8,5 5,6 5,5 8,7	**Saída** 6,4
Entrada 7 8 10 4 6,5 9,5 8 6,7 9,5 6,2	**Saída** 7,5
Entrada 10 8 7 6 9 8,9 9 6 8 10	**Saída** 8,1

E.7 Lista7

Média Turma

Faça um programa que leia diversos valores de notas de alunos e informe a média aritmética da turma, ou seja a soma das notas dividido pela quantidade de alunos.

Entrada

A entrada consiste vários números que representam notas dos estudantes, a entrada termina quando for informado um valor de nota igual -1(menos um).

Saída

A saída do seu programa deve ser um valor que corresponde à média da turma.

Exemplos

Entrada 0 0 -1	**Saída** 0,0
Entrada 10 8 -1	**Saída** 7,0
Entrada 8 6 9 -1	**Saída** 7,6
Entrada 10 8 7 6 -1	**Saída** 7,7
Entrada 9 4 3 5 2 4 -1	**Saída** 4,5

Vice-líder

A OBI (Organização de Bocha Internacional) é responsável por organizar a competição mundial de bocha. Infelizmente esse esporte não é muito popular, e numa tentativa de aumentar a sua popularidade, ficou decidido que seriam chamados, para a Grande Final Mundial, o campeão e o vice-campeão de cada sede nacional, ao invés de apenas o primeiro lugar.

Tumbólia é um país pequeno que já havia realizado a sua competição nacional quando a nova regra foi instituída, e o comitê local não armazenou quem foi o segundo classificado. Felizmente eles armazenaram a pontuação de todos competidores - que foram apenas três, devido ao tamanho diminuto do país. Sabe-se também que as pontuações de todos jogadores foram diferentes, de forma que não ocorreu empate entre nenhum deles.

Resta agora descobrir quem foi o vice-campeão e para isso o comitê precisa de ajuda.

Entrada

A primeira e única linha da entrada consiste de três inteiros separados por espaços, *A*, *B* e *C*, as pontuações dos 3 competidores.

Saída

Imprima uma única linha na saída, contendo apenas um número inteiro, a pontuação do vice-campeão.

Exemplos

Entrada 1 2 3	**Saída** 2
Entrada 1 0 7	**Saída** 1
Entrada 10 5 9	**Saída** 9
Entrada 4 5 6	**Saída** 5

Fonte: Tarefa da OBI2012, Modalidade Programação Nível 1, Fase 1

Receita de bolo

João deseja fazer bolos para seus amigos, usando uma receita que indica que devem ser usadas 2 xícaras de farinha de trigo, 3 ovos e 5 colheres de sopa de leite. Em casa ele tem A xícaras de farinha de trigo, B ovos e C colheres de sopa de leite. João não tem muita prática com a cozinha, e portanto ele só se arriscará a fazer medidas exatas da receita de bolo (por exemplo, se ele tiver material suficiente para fazer mais do que 2 e menos do que 3 bolos, ele fará somente 2 bolos). Sabendo disto, ajude João escrevendo um programa que determine qual a quantidade máxima de bolos que ele consegue fazer.

Entrada

A entrada é dada em uma única linha, que contém três números inteiros A, B e C, indicando respectivamente o número de xícaras de farinha de trigo, o número de ovos e o número de colheres de sopa de leite que João tem em casa.

Saída

Seu programa deve imprimir uma única linha, contendo um único inteiro, a quantidade máxima de bolos que João consegue fazer.

Exemplos

Entrada 0 0 0	**Saída** 0
Entrada 4 6 9	**Saída** 1
Entrada 4 6 10	**Saída** 2
Entrada 8 9 15	**Saída** 3

OBI

O principal prêmio da Olimpíada Brasileira de Informática é o convite para os cursos de programação oferecidos no Instituto de Computação da Unicamp, com todas as despesas pagas pela Fundação Carlos Chagas, patrocinadora da OBI. São convidados apenas os competidores que atingem um certo número mínimo de pontos, consideradas as duas fases de provas. Você foi contratado pela Coordenação da OBI para fazer um programa que, dados os números de pontos obtidos por cada competidor em cada uma das fases, e o número mínimo de pontos para ser convidado, determine quantos competidores serão convidados para o curso na Unicamp. Você deve considerar que todos os competidores participaram das duas fases; o total de pontos de um competidor é a soma dos pontos obtidos nas duas fases.

Por exemplo, se a pontuação mínima para ser convidado é 435 pontos, um competidor que tenha obtido 200 pontos na primeira fase e 235 pontos na segunda fase será convidado para o curso na Unicamp. Já um competidor que tenha obtido 200 pontos na primeira fase e 234 pontos na segunda fase não será convidado.

Entrada

A primeira linha da entrada contém dois números inteiros N e P, representando respectivamente o número de competidores e o número mínimo de pontos para ser convidado. Cada uma das N linhas seguintes contém dois números inteiros X e Y indicando a pontuação de um competidor em cada uma das fases.

Saída

Seu programa deve produzir uma única linha contendo um único inteiro, indicando o número de competidores que serão convidados a participar do curso na Unicamp.

Exemplos

Entrada	**Saída**
0 0	0
Entrada	**Saída**
1 0 1 1	1
Entrada	**Saída**
2 200 90 25 123 37	0
Entrada	**Saída**
3 100 50 50 100 0 49 50	2
Entrada	**Saída**
4 235 100 134 0 0 200 200 150 150	2

Fonte: OBI 2008, Fase 1, Nível Júnior.

Fibonacci

A sequência de Fibonacci é uma das sequências de números bastante famosas. Ela inicia da seguinte forma:

1, 1, 2, 3, 5, 8, 13, 21, ...

Sendo que:

- 2 é a soma (1+1)
- 3 é a soma (1+2)
- 5 é a soma (2+3)
- 8 é a soma (3+5)
- etc...

Construa um programa que gere uma sequência de Fibonacci de uma quantidade N de elementos.

Entrada

A entrada consiste em um número inteiro N que indica quantidade de elementos da sequência.

Saída

Seu programa deve imprimir os elementos separados por espaço em uma única linha.

Exemplos

Entrada 1	**Saída** 1
Entrada 2	**Saída** 1 1
Entrada 3	**Saída** 1 1 2
Entrada 6	**Saída** 1 1 2 3 5 8
Entrada 10	**Saída** 1 1 2 3 5 8 13 21 33 55
Entrada 12	**Saída** 1 1 2 3 5 8 13 21 34 55 89 144