UNIVERSIDADE FEDERAL DO PARANÁ

MARIANGELA DE OLIVEIRA GOMES SETTI

O Processo de Discretização do Raciocínio Matemático na Tradução para o Raciocínio Computacional: Um Estudo de Caso no Ensino/Aprendizagem de Algoritmos

CURITIBA

2009

MARIANGELA DE OLIVEIRA GOMES SETTI

O Processo de Discretização do Raciocínio Matemático na Tradução para o Raciocínio Computacional: Um Estudo de Caso no Ensino/Aprendizagem de Algoritmos

Tese apresentada ao Curso de Pós-Graduação em Educação, Setor de Educação, Universidade Federal do Paraná, como requisito parcial para a obtenção do grau de Doutora em Educação, na linha de pesquisa Educação Matemática.

Orientador: Prof. Dr. José Carlos Cifuentes

CURITIBA

2009

Sumário

Lista de Quadros

Lista de Figuras

“Só sabemos com exatidão quando sabemos pouco;
à medida que vamos adquirindo conhecimentos,
instala-se a dúvida.”
(Johann Goethe)

Agradecimentos

Para agradecer a todos aqueles que contribuíram com o desenvolvimento deste trabalho, seria preciso muito mais do que algumas linhas, são tantas as pessoas que devo agradecer, que é preciso muito cuidado no momento de nominá-las.

Primeiro agradeço a Deus, por iluminar e abençoar meus caminhos em mais esta etapa da minha vida.

A meu marido, João Antônio Palma Setti pelo amor, cumplicidade e compreensão em todo este período, e a meu filho, Matheus Gomes Setti, por ser a motivação de todo meu esforço.

Agradeço também a minha mãe, que sempre foi exemplo de força e coragem.

Ao Prof. Dr. José Carlos Cifuentes, pela paciência, compreensão, concordâncias e discordâncias, como não poderia deixar de ser um processo de orientação, acima de tudo, agradeço pelo carinho e amizade, que com certeza, começaram com o desenvolvimento deste trabalho, e à sua esposa Dra. Blanca Beatriz Diaz Alva, pelo incentivo e amizade.

Aos professores membros da banca de defesa: Dr. Ítalo Modesto Dutra (UFRGS), Dr[a]. Sonia Barbosa Igliori (PUC-SP), Dr[a]. Célia Finck Brandt (UEPG) e Dr. Gustavo Giménez Lugo (UTFPR) pelas contribuições dadas.

A todos os amigos, professores e funcionários do Programa de Pós-Graduação em Educação (PPGE), da Universidade Federal do Paraná, que de alguma, contribuíram para o desenvolvimento deste trabalho.

À Leônia Gabardo Negrelli e Elisângela Campos pela colaboração, sugestões e tantas discussões que, sem dúvida, tiveram uma grande importância para a melhor conclusão deste trabalho.

A todos os colegas da Universidade Tecnológica Federal do Paraná (UTFPR), especialmente aos colegas do Departamento de Informática, que contribuíram para a conclusão deste trabalho.

Aos alunos do Curso de Tecnologia em Informática da UTFPR, sem os quais, grande parte deste trabalho não poderia ter sido realizado.

Finalmente, agradeço aos meus amigos e à minha família, que são a melhor coisa da

minha vida!

Resumo

Este trabalho aborda a aprendizagem e o ensino de introdução à programação, cujo conteúdo central é a elaboração de algoritmos computacionais. Observa-se que muitos dos alunos, embora detenham conhecimentos matemáticos oriundos do Ensino Médio, para resolver problemas matemáticos encontram dificuldades na passagem do raciocínio matemático para o correspondente computacional. Para realizar esta passagem, é necessário utilizar os conhecimentos adquiridos previamente com um novo formato, devido ao processo de discretização necessário para transformar o raciocínio matemático no correspondente computacional. Para conceber esta discretização, os conhecimentos matemáticos estabelecidos irão sofrer uma "ruptura epistemológica", pois se trata de uma mudança na forma de compreender um conhecimento. Além disso, para elaborar um algoritmo, utilizam-se registros de representação semiótica diferentes daqueles utilizados pela linguagem natural, e pela linguagem matemática, o que vem a ser mais um fator produtor de obstáculo. Para identificar as principais dificuldades dos alunos nesta fase, realizamos uma pesquisa empírica, por meio de um estudo de caso com múltiplos casos, concluindo que elas estão relacionadas às estruturas de repetição, necessárias para realizar o processo de discretização, bem como às diferenças entre os registros de representação semiótica de partida (linguagem natural) e os de chegada (linguagem computacional).

Palavras-Chave: Algoritmos, Discretização, Obstáculos Epistemológicos e Registros de Representação Semiótica.

Abstract

This work deals with learning and teaching of programming introduction, which central focus is the development of computational algorithms. It is observed that many of the students, although having mathematical skills acquired in high school, have difficulties to solve mathematical problems when transposing the mathematical reasoning to its corresponding computational thinking. For this transition to be made, it is necessary to use the knowledge acquired previously within a new arrangement, due to the process of discretization required to change the mathematical reasoning into its computational correlative. To conceive such discretization, the established mathematical knowledge will undergo an "epistemological rupture", because it is a change in the way of understanding knowledge. Moreover, to develop an algorithm, records of semiotic representation, other than those used by natural language and mathematic language, are employed, becoming an additional hindrance-producing factor. To identify the major difficulties of the students at this stage, we conducted an empirical research through a case study with multiple cases, concluding that they are related to the structures of repetition, necessary to accomplish the process of discretization, as well as to the differences between the records of semiotic representation at the start (natural language) and those at the end (computer language).

Key Words: Algorithms, Discretization, Epistemological Obstacles and Semiotics Representation Records.

Apresentação

Desenvolvemos este estudo como trabalho de conclusão do Curso de Doutorado em Educação, da Universidade Federal do Paraná (UFPR), na linha de pesquisa *Educação Matemática*. Nele abordamos a aprendizagem e o ensino das disciplinas de Introdução à Programação, cujo conteúdo central é a elaboração de algoritmos computacionais. Nestas disciplinas, muitos dos alunos, embora detenham conhecimentos matemáticos oriundos do Ensino Médio, para resolver problemas matemáticos, encontram dificuldades na passagem do raciocínio intuitivo, ainda que matemático, para o formal computacional.

Entendemos como problemas matemáticos, situações em que é possível extrair do fenômeno em questão aspectos como simetria, regularidade, homogeneidade e uniformidade. Para elaborar um algoritmo que resolva um problema proposto, o aluno deve ser capaz de identificar estes aspectos, o que nem sempre é trivial.

Não se trata apenas de saber como resolver um problema, baseado no ferramental trazido do Ensino Médio, trata-se de utilizar um conhecimento adquirido para solucionar um problema com um novo formato, qual seja, conceber a solução computacional correspondente à solução matemática.

Para superar esta diferença, os conhecimentos matemáticos estabelecidos irão sofrer uma “ruptura epistemológica”, pois se trata de uma mudança na forma de compreender um conhecimento, considerando que nos processos matemáticos trabalhados até o final do Ensino Médio, a matemática utilizada tem um caráter contínuo, enquanto as soluções computacionais têm um caráter discreto, em que o fator “tempo” deve ser considerado.

Além da mudança na forma de compreender o conhecimento, para elaborar um algoritmo utilizam-se registros de representação semiótica diferentes daqueles utilizados pela linguagem natural e pela linguagem matemática, mais uma possível fonte de obstáculos epistemológicos.

Este trabalho está organizado da seguinte maneira:

O Capítulo 1 faz a introdução do problema investigado, apresentando também nossas hipóteses de trabalho e os objetivos a serem alcançados.

O Capítulo 2 faz um histórico do conceito de algoritmo, abordando alguns acontecimentos na história da Matemática, que tiveram papel fundamental para a concepção e

o desenvolvimento da Ciência da Computação. O ponto central é a teoria que possibilitou a criação do algoritmo computacional.

No Capítulo 3 é apresentada a base teórica para este trabalho, incluindo assuntos ligados à Educação Matemática, bem como à Educação e Informática. Discutimos aspectos relacionados à matemática discreta, ao conceito de obstáculo epistemológico e à teoria de registros de representação semiótica, fazendo uma conexão com o processo de ensino e aprendizagem de algoritmos.

No Capítulo 4 abordamos alguns aspectos relevantes do processo de aprendizagem de Algoritmos. Primeiro, apresentamos as formas de representação para um algoritmo; em seguida, as estruturas disponíveis para a elaboração de um algoritmo; por fim, estabelecemos uma relação entre a aprendizagem de algoritmos e a aprendizagem de matemática.

O Capítulo 5 apresenta o estado da arte no ensino de algoritmos, descrevendo brevemente algumas metodologias de ensino, utilizadas por pesquisadores para o ensino de disciplinas que abordam a introdução à programação, mostrando as vantagens e as desvantagens das mesmas.

O Capítulo 6 descreve os procedimentos metodológicos adotados para realizar a pesquisa de campo, parte deste trabalho.

No Capítulo 7 apresentamos os casos analisados em nosso estudo de caso, composto por três casos: o primeiro, aborda a aprendizagem da estrutura de repetição, utilizada no processo de discretização do raciocínio, necessário à passagem do raciocínio matemático para o correspondente computacional; o segundo, aborda os registros de representação semiótica utilizados para representar um algoritmo; e o terceiro, faz uma comparação entre a elaboração do raciocínio matemático e do raciocínio computacional.

Finalmente, o Capítulo 8 apresenta as conclusões, considerações finais e alguns apontamentos para trabalhos futuros.

CAPÍTULO 1

O Problema de Investigação

Os recursos tecnológicos, entre eles os computadores, estão popularizando-se rapidamente nas mais variadas áreas e atividades, notadamente no ambiente escolar, nos seus diversos níveis. O computador pode ser utilizado para realizar inúmeras tarefas, porém, para que desempenhe adequadamente sua função, é necessário que receba instruções rigorosamente detalhadas e precisas.

Para que o processamento dessas instruções ocorra corretamente, deve-se estabelecer qual a lógica que o computador deve utilizar. Na Ciência da Computação, essa lógica chama-se *lógica de programação* ou *lógica de construção de algoritmos,* e consiste em conceber processos de raciocínio, por meio de simbolizações formais, para a programação de computadores. Observe-se que "lógica", uma vez usada em relação ao funcionamento de um computador, está sendo entendida de forma rígida. Assim, não é qualquer tipo de argumentação que constitui uma lógica.

A palavra raciocínio está intimamente ligada aos processos mentais do ser humano. Sua relação com os processos computacionais, exige um estudo aprofundado para sua melhor compreensão.

Os seres humanos têm a capacidade de expressar seu raciocínio por meio da palavra falada ou escrita, que, por sua vez, baseia-se em um determinado idioma, que segue uma série de padrões definidos por uma gramática. Um mesmo raciocínio pode ser expresso em qualquer um dos inúmeros idiomas existentes; todavia, continuará representando o mesmo raciocínio, usando apenas outra convenção.

O mesmo ocorre com a lógica de programação, que pode ser representada em qualquer uma das inúmeras linguagens de programação existentes. Estas, por sua vez, são atreladas a uma grande diversidade de detalhes computacionais, que pouco tem a ver com o raciocínio envolvido. Por conta disso, um dos instrumentos mais utilizados para expressar a lógica de programação é o *algoritmo,* porque uma vez concebida uma solução algorítmica para um problema, esta pode ser traduzida para qualquer linguagem de programação e ser agregada às funcionalidades disponíveis nos diversos ambientes existentes.

Quando elaboramos um algoritmo, devemos especificar ações claras e precisas, que, a partir de um estado inicial, após um período de tempo finito, produzam um estado final previsível e bem definido. Isso significa que o algoritmo fixa um padrão de comportamento a ser seguido, uma norma de execução a ser trilhada, com vistas a alcançar, como resultado final, a solução de um problema, garantindo que, sempre que executado, sob as mesmas condições, produza o mesmo resultado. Este é o caráter lógico do algoritmo.

A primeira vez que um algoritmo foi escrito para um computador foi em 1842, por Ada Byron, para a máquina desenvolvida por Charles Babbage. Porém, como Babbage não concluiu sua máquina, este algoritmo nunca foi implementado. Entretanto, a idéia de algoritmo habitava a consciência dos matemáticos do mundo todo desde o século XVII. Todavia, apenas na década de 30 essa idéia teve, dentre outras, quatro definições descritivas diferentes dadas, no âmbito da lógica moderna, por Gödel, Church, Post e Turing (SAGASTUME, 2003).

Para o momento adotaremos provisoriamente a definição informal de algoritmo, sugerida acima, como sendo um conjunto de comandos que, a partir de um estado inicial, após um período de tempo finito, produz um estado final previsível e bem definido. Alguns autores atribuem denominações distintas para esta mesma definição, como procedimento efetivo, procedimento mecânico e tarefa. Neste trabalho, estas denominações são consideradas como sinônimos. Na seção 2.1, o termo ‘algoritmo’ será descriminado com o formalismo pertinente.

Nos anos 70, as atividades de pesquisa conduziram ao reconhecimento da programação de computadores como uma verdadeira disciplina, cujo conhecimento é fundamental para o sucesso de muitos projetos ligados às mais variadas áreas do saber (WIRTH, 1986).

Segundo Wirth (1986), a metodologia utilizada no ensino de algoritmos, na área de Informática e cursos afins, até o final dos anos 90 era baseada, na maioria dos casos, na programação estruturada, proposta por Dahl, Dijkstra e Hoare, aproximadamente em 1970. Essa metodologia tem como objetivo minimizar a complexidade dos programas computacionais.

A consolidação da metodologia de orientação a objetos, no final dos anos 1990, levou, após muitas discussões, diversos grupos a adotarem-na para o ensino de Introdução à Programação, como descrito no capítulo 5 deste trabalho. A discussão sobre qual metodologia

é a mais adequada para ensinar alunos iniciantes persiste atualmente; não há consenso sobre o melhor caminho, se começar com a metodologia estruturada ou com a orientação à objetos. Considerando que as diferenças entre ambas se intensificam apenas no momento de codificar o algoritmo por meio de uma linguagem de programação, a utilização de uma ou outra não altera nosso objeto de estudo. Cabe destacar que a lógica envolvida na concepção do algoritmo é semelhante nos dois casos, assim como as dificuldades encontradas pelos alunos.

Considerando que a programação abrange uma grande variedade de atividades intelectuais complexas, Wirth não acreditava que esse processo pudesse ser condensado em uma espécie de receita didática e nós concordamos com ele. Este trabalho não tem a pretensão de chegar à referida receita, mas, sim, buscar subsídios para nortear o ensino de algoritmos.

1.1 – O Contexto do Ensino - Aprendizagem

O ensino e a aprendizagem de algoritmos têm sido objetos de estudo de diversos grupos de pesquisa. Uma das razões disso é que a eficácia dos métodos utilizados pelos educadores, até o presente momento, tem estado muito aquém do ideal. Pesquisando-se as possíveis causas dos baixos índices de aproveitamento, observa-se que a maioria dos alunos encontra dificuldades na passagem do raciocínio intuitivo, ainda que matemático, para o formal computacional. A construção de algoritmos exige do aluno capacidades de abstração, de análise e síntese, de raciocínio combinatório, e os cursos, em geral, carecem de meios eficazes para que eles trabalhem suas idéias e construam o seu próprio conhecimento (SETTI e CIFUENTES, 2006).

Verificamos, nas seguintes bibliografias utilizadas para o ensino das disciplinas que abordam o conteúdo de algoritmos: (FARRER, 1999), (FORBELLONE, 2000), (GERSTING, 1995), (SALVETTI, 1999), (ASCENCIO, 2002), (WIRTH, 1989), (KNUTH, 1968), que os exercícios utilizados para induzir o aprendizado da lógica de programação são quase sempre problemas matemáticos.

Podemos notar, nas disciplinas de introdução à programação, ministradas no início de diversos cursos de graduação, que, apesar de os alunos terem concluído o Ensino Médio, apresentam muitas dificuldades em conceber uma solução computacional clássica, correspondente aos problemas matemáticos trabalhados anteriormente.

Segundo Newell (1972), um problema é uma situação na qual um indivíduo deseja

fazer algo, porém desconhece o caminho das ações necessárias para concretizar a sua ação. Já para Chi *et al*. (1988), problema é uma situação na qual um indivíduo atua com o propósito de alcançar uma meta, utilizando para tal alguma estratégia em particular. É comum confundir a palavra ‘problema’ com ‘exercício’, sendo que, muitas vezes, eles são utilizados como equivalentes. Deve-se observar que o exercício envolve mera aplicação de conteúdos ministrados, enquanto o problema necessariamente envolve invenção e/ou criação significativa.

Não temos a intenção de utilizar uma definição rigorosa de problema, porém entendemos como “problemas matemáticos”, situações em que é possível extrair do fenômeno em questão aspectos como simetria, uniformidade, regularidade e periodicidade.

Por exemplo, analisando o enunciado:

Um certo cidadão *A* tem 1,5m de altura e cresce 2cm por ano, enquanto um outro cidadão *B* tem 1,10m e cresce 3cm por ano. Construa um algoritmo que calcule e imprima quantos anos serão necessários para que o cidadão *B* seja maior que o cidadão *A* (ASCENCIO, 2002).

O que torna este enunciado um problema matemático, não é a presença dos números, mas o fato de aumentar um valor constante a cada ano. Para elaborar um algoritmo que resolva a situação proposta, o aluno deve ser capaz de identificar esses aspectos, neste caso, a regularidade, o que nem sempre é trivial.

Adotaremos, neste trabalho, como objetos para a formulação de algoritmos um subconjunto dos problemas matemáticos, aqueles que são também problemas algorítmicos, em particular os “problemas iterativos”. Nessa classe de problemas, o enunciado apresenta uma situação para cuja resolução é necessária a utilização de estruturas de decisão e/ou repetição. No exemplo anterior, tem-se a seguinte solução algorítmica:

```
inicio          // início do algoritmo em portugol
inteiro A, B, Anos;
Anos ← 0;
A ← 1,5;  B ← 1,1;
enquanto (A ≥ B)
inicio
```

```
        A ← A + 0,02;
        B ← B + 0,03;
        Anos ← Anos + 1;
fim_enquanto
Escreva (“O número de anos para que B seja maior do que A é ”,  Anos);
fim.
```

Observe, que muitas vezes, pode ser confundido o “resolver o problema” com “elaborar um algoritmo que o resolva” e que o computador seja capaz de implementar.

Uma das dificuldades em distinguir esses dois tipos de soluções está na diferença entre o pensar matematicamente[1] e o pensar computacionalmente, ou entre o pensamento contínuo e o sequencial discreto. Ensinar o aluno a pensar matematicamente, ou computacionalmente, não é apenas uma questão de mudança de linguagem, torna-se necessário haver uma alteração da forma de pensar.

Não se trata apenas de saber como resolver um problema, baseado no ferramental trazido do Ensino Médio, trata-se de utilizar um conhecimento adquirido para solucionar um problema com um novo formato, qual seja, conceber a solução computacional correspondente à solução matemática.

Consideramos que a matemática escolar, ou seja, a matemática ministrada na escola, pode ser dividida em duas áreas, a saber, a matemática do contínuo, cujo paradigma pode ser colocado em disciplinas como o Cálculo Diferencial e Integral e a matemática do discreto ou matemática discreta, que se desenvolveu a partir do séc. XX e abrange principalmente a matemática combinatória e estatística, teoria dos grafos e jogos. Para trabalhar com a matemática discreta, é necessário utilizarmos outras formas de pensar, próprias dos problemas que podem ser implementados computacionalmente.

Apesar do avanço computacional nas últimas décadas, a matemática escolar ainda não incorporou a forma própria que o pensamento algorítmico exige. O que tem ocorrido é a utilização de ferramentas computacionais, como suporte ao ensino em diversas áreas. Essa é uma dificuldade que os alunos enfrentam ao se depararem com esse assunto no ensino superior, o que torna a matemática escolar um obstáculo, inclusive de caráter epistemológico, à aprendizagem de algoritmos.

---

[1] Segundo Schoenfeld (1998, p. 59), significa “(a) ver o mundo de um ponto de vista matemático (tendo predilecção por matematizar: modelar, simbolizar, abstrair e aplicar idéias matemáticas a uma larga gama de situações) e, (b) ter os instrumentos para tirar proveito para matematizar com sucesso.”

Segundo Ernest (1998), dentre as principais mudanças que ocorreram acerca da concepção do conhecimento, está o reconhecimento da distinção entre os conhecimentos *explícito* e *tácito*. No entendimento de Ernest (1998), um conhecimento matemático explícito é aquele que pode ser adquirido por meio da linguagem, mesmo que informal, ou de demonstrações, como, por exemplo, o conteúdo do teorema de Pitágoras. Por outro lado, um conhecimento matemático tácito é aquele adquirido por meio da ação ou da experiência e que não pode ser totalmente explicitado por meio da linguagem proposicional.

Podemos entender o conhecimento matemático elementar como sendo um tipo de conhecimento tácito. Porém, o conhecimento tácito não se manifesta apenas na mera aplicação mecânica de fórmulas ou procedimentos, ele exige uma certa compreensão, o que, para SCHOENFELD (1992), é o pensar matematicamente. Segundo o autor, para uma pessoa aprender Matemática não basta que ela se aproprie e faça uso de ferramentas matemáticas, ela precisa desenvolver algo mais, que iremos chamar de “um pensar matemático”.

Parafraseando SCHOENFELD (1992), para aprender a elaborar um algoritmo não basta se apropriar e fazer uso de ferramentas computacionais, como fluxograma e pseudocódigo, é preciso desenvolver um pensar “computacional”.

Nos processos envolvidos, ligados à aprendizagem de algoritmos para concretizar esse pensar, observa-se que, além da possibilidade de existirem obstáculos de tipo epistemológico, há uma conversão entre registros de representação semiótica, pois para elaborar um algoritmo, utilizam-se registros de representação semiótica diferentes daqueles utilizados pela linguagem natural e pela linguagem matemática. Para enfrentar a problemática oriunda dessa operação cognitiva de conversão, que pode colocar em cena o fenômeno da não congruência semântica, utilizamos a teoria de representações semióticas, conforme Duval (1995). As situações problema escolhidas foram aqueles que envolvem processos condicionais e iterativos.

Desde 1998 temos ministrado aulas na Universidade Tecnológica Federal do Paraná UTFPR-PR, até 2005 Centro Federal de Educação Tecnológica do Paraná (CEFET-PR), tanto para os cursos de Engenharia, das disciplinas *Introdução à Programação* e *Programação Avançada*, quanto para o Curso Superior de Tecnologia em Informática, da disciplina *Lógica Aplicada*, atualmente chamada de *Lógica de Programação*. Esta última é uma disciplina de introdução à lógica matemática e algoritmos. Considerando as dificuldades enfrentadas,

enquanto estudante do curso de Bacharelado em Ciência da Computação, temos procurado, na medida do possível, organizar a disciplina de forma a minimizá-las.

Nos últimos anos, alguns questionamentos têm surgido, por conta das nossas preocupações com a aprendizagem de algoritmos, e a busca de respostas para alguns desses questionamentos é um dos fatores que nos motivaram à realização deste estudo.

Observamos que as dificuldades dos alunos em relação às novas formas de pensamento começam a aparecer quando surge a necessidade de trabalhar com processos repetitivos iterativos, que envolvem a identificação das regularidades do problema que se quer solucionar e a consequente discretização do raciocínio, necessária à transformação deste no correspondente computacional. Normalmente, aqueles alunos que não apresentam dificuldade com esse conteúdo, também não apresentam outras dificuldades significativas; porém, aqueles que não compreendem esses processos, tendem a ficar desmotivados, o que faz com que estas dificuldades se agravem ao longo do curso.

Tal situação é ilustrada por Berlinski (2002, p. 51), ao mencionar que,

> Expressões em linguagem comum, como por exemplo, tomar, repetir, imaginar, comparar ou descobrir, indicam que algo seja feito. Porém ao pretender que a soma seja feita indefinidamente, a mente de repente patina sobre gelo onde antes havia um caminho sólido e escorrega sem parar e sem ponto de apoio.

O ensino superior em instituições públicas no Brasil passa, de modo geral, por um problema sério de evasão. Estima-se que cerca de 230 mil acadêmicos abandonem os seus cursos durante o decorrer dos mesmos, o que corresponde a 64 estudantes concluintes de 100 ingressantes (PORTO, 2003).

A Comissão de Especialistas de Ensino de Computação e Informática, do MEC/SESu, fez uma análise estatística, com base nos dados do INEP (Instituto Nacional de Estudos e Pesquisas Educacionais Anísio Teixeira), fornecidos pelas Instituições de Ensino Superior para os Censos de 2001 a 2006, disponível em (MEC, 2009). Nesta análise, verifica-se que a relação entre alunos ingressantes e concluintes é de aproximadamente 15%, estes indicadores confirmam a necessidade de alterações no Ensino Superior, notadamente nas áreas de Ciências Exatas e Tecnológicas.

Com relação ao ensino de algoritmos, observa-se que os métodos utilizados variam muito, dependem, entre outros, do professor, do curso e da escola. A aplicação do

conhecimento se dá de modo geral via lápis e papel, por meio de uma linguagem de programação, pseudocódigo (portugol), ou da utilização de ferramentas para a compilação de algoritmos, sejam esses escritos em linguagem de programação ou em pseudocódigo. Ainda assim, há diferenças entre as escolas, não só quanto à linguagem de programação, mas também, quanto ao compilador específico utilizado (GUIMARÃES, 1995).

A Sociedade Brasileira de Computação promove congressos, listas de discussões e grupos de trabalho em diversas áreas da Informática. Anualmente, no âmbito do Congresso da Sociedade Brasileira de Computação, entre os eventos relevantes para este trabalho estão o *Workshop* sobre Educação e Informática e o Curso de Qualidade de Cursos de Graduação da Área de Computação e Informática. Nesses dois eventos discute-se o currículo dos cursos de Informática, metodologias de ensino, formação dos professores e são apresentados estudos de casos em diversos ramos dos cursos.

No WEI (2008), foram apresentados diversos relatos de experiências sobre o ensino de introdução à programação (algoritmos), em cursos de computação. Dois pontos merecem destaque analisando os relatos, o primeiro é que não houve aumento significativo em relação ao número de alunos aprovados; o segundo é que aqueles que atingiram êxito terminaram a disciplina com mais maturidade e avançaram mais do que aqueles que foram formados utilizando metodologia tradicional. Ou seja, a classe de alunos com mais aptidão ganhou em qualidade e tecnologia; porém, a classe daqueles com mais dificuldade continuou à margem do aprendizado.

Por meio de pesquisa bibliográfica inicial, pudemos constatar que a aprendizagem de algoritmos tem passado historicamente pelos mesmos problemas que diversas outras disciplinas da área de Ciências Exatas, especialmente a Matemática. Nesses casos, há que se considerar também o número elevado de reprovações em disciplinas que abordam esses conteúdos, o que, muitas vezes, provoca evasão dos referidos cursos.

Ao conseguirmos identificar as dificuldades contidas neste processo, a evasão dos cursos que utilizam estes conteúdos poderia ser minimizada, pois os alunos não seriam desencorajados num período delicado de adaptação ao ingresso em curso superior, em que já enfrentam vários problemas. De fato, parte dessa adaptação consiste em superar alguns obstáculos de tipo epistemológico ao aprendizado do “novo” pensamento computacional, notoriamente o conhecimento matemático tácito que o aluno traz do Ensino Médio, em que

não são incentivadas as capacidades de raciocínio que o pensamento algorítmico (discreto) precisa.

A partir desta discussão inicial, apresentamos a seguir, algumas questões que norteiam o presente trabalho de pesquisa.

1.2 – Questões Norteadoras e Considerações Preliminares

- Quais são as dificuldades na aprendizagem da resolução dos problemas iterativos na disciplina de algoritmos?
- Essas dificuldades estão ligadas a fenômenos de representação semiótica?
- Qual o caráter epistemológico dessas dificuldades?
- Em que medida o pensamento algorítmico é diferente do pensamento matemático, e como essa diferença acentua essas dificuldades?

Uma das intenções deste trabalho é estender a discussão sobre os obstáculos epistemológicos (e outros), à aprendizagem de algoritmos, considerando que a situação em questão, está ligada à forma de pensar sobre a solução de um problema e não sobre o próprio problema. Com isso, espera-se identificar os motivos que levam os alunos a não conseguirem transformar os processos matemáticos que aprenderam na escola em algoritmos computacionais, isto é, transformar o pensamento matemático em pensamento computacional.

Também, pretende-se fazer uma análise dos diferentes registros de representações semióticas utilizados no ensino-aprendizagem de algoritmos.

Uma vez identificadas as principais dificuldades na aprendizagem de algoritmos, poderíamos utilizar uma metodologia de ensino que auxilie a superá-las, o que tornaria possível reduzir o número de alunos que não conseguem obter êxito na sua aprendizagem.

Considerando que a UTFPR atua tanto no nível técnico, quanto na graduação e pós-graduação, principalmente nas áreas de Ciências Exatas e Tecnologia, embora não somente, um estudo sobre as dificuldades da aprendizagem de algoritmos poderá trazer consequências relevantes para seu ensino. Possivelmente, os resultados deste trabalho poderão ser estendidos às demais instituições que atuam em área semelhante, tendo em vista o número significativo delas que apresentam o mesmo tipo de problema.

1.3 – Pressupostos

Neste trabalho adotamos os seguintes pressupostos:

- Elaborar um algoritmo exige uma alteração de forma de pensar, conceber, nos processos de raciocínio matemático elementar.
- Os conhecimentos matemáticos estabelecidos irão sofrer “rupturas” nessa alteração e, considerando que se trata de formas de compreender, pode-se dizer que acontecem ‘rupturas epistemológicas’.
- As rupturas epistemológicas estão ligadas ao conceito de ‘obstáculo epistemológico’ e envolvem mudanças na forma de compreender um conhecimento. De fato, nos processos matemáticos trabalhados até o final do Ensino Médio, a Matemática utilizada tem um caráter notadamente contínuo e as soluções computacionais têm um caráter discreto, em que o tempo e o universo finitos devem ser considerados.
- As formas de representação utilizadas na elaboração dos algoritmos podem ser mais uma fonte de obstáculo, pois mobilizam registros de representação semiótica diferentes daqueles utilizados pela linguagem natural.

Com o intuito de lançar luz às nossas idéias, buscamos informações sobre os assuntos relevantes ao tema em questão em diversas áreas, analisando diversos trabalhos tanto da área de Educação Matemática, quanto da área de Informática, bem como da área de Educação e Informática. Neste levantamento, pesquisamos dissertações de mestrado e teses de doutorado (COSTELLOE, 2004ª, 2004b), (GUIMARÃES, 1995), (BROLEZZI, 1996), (FRIEDMANN, 2003), (TRINDADE, 1996), disponíveis no sítio da Capes e de diversas universidades brasileiras e estrangeiras, entre outras, listadas nas referências bibliográficas ao final deste trabalho.

Considerando o que diz Bruyne (1991) sobre a epistemologia estabelecer as condições de objetividade dos conhecimentos científicos, dos modos de observação e de experimentação, examinando igualmente as relações que as ciências estabelecem entre as teorias e os fatos, escolhemos como base teórica as idéias de Bachelard (1938), Brousseau (1986) e Duval (1995). Esta escolha se deu em função da noção de ‘obstáculo epistemológico’ ter sido definida por Bachelard, e aplicada ao ensino de Matemática por Brousseau. Já Duval, elaborou uma teoria sobre aprendizagem, com base em registros de representação semiótica.

Segundo Schubring (2002), Bachelard estabeleceu a sua concepção de obstáculo

epistemológico a fim de investigar o pensamento científico, tendo em vista refletir sobre a história da ciência. Seus trabalhos fazem uma análise da formação do conhecimento. Neste trabalho, consideramos que os obstáculos no ensino de algoritmos, podem estar relacionados à formação e à adaptação de um conhecimento, consequentemente podem envolver também rupturas epistemológicas, pois para pensar computacionalmente é necessário romper com o pensamento intuitivo para sua devida formalização, especialmente no caso das estruturas de repetição, que será um dos nossos focos nesta pesquisa, bem como outras estruturas próprias do pensamento computacional.

Schubring (2002) não partilha da opinião de Brousseau com relação ao fato dos obstáculos epistemológicos serem incontornáveis, pois se assim fosse, como se dariam os progressos científicos? Entretanto, ele considera que diversos aspectos na teoria criada por Brousseau podem contribuir para o processo de identificação dos obstáculos.

Glaeser (1981), por outro lado, propôs a interpretação de "obstáculo" como "dificuldade", e Sierpinska (1985) construiu uma abordagem própria, em que o papel dos obstáculos epistemológicos é o de auxiliar na melhor compreensão da Matemática, o que poderíamos adaptar para o caso dos algoritmos.

Em 1988, em Quebec, aconteceu um simpósio internacional: *Obstacles et conflits cognitifs* (Obstáculos em Conflitos Cognitivos), (SCHUBRING, 2002), mas, mesmo com a participação de muitos especialistas, não foi possível responder a perguntas como, por exemplo: como reconhecer um obstáculo de caráter epistemológico? Ou ainda, é possível evitá-los no ensino/aprendizagem? Como ultrapassá-los?

A escolha da área de Educação Matemática para inserção deste trabalho se deu em função da proximidade de ambos, considerando que a Matemática é amplamente utilizada para o ensino de algoritmos computacionais e que eles, no fundo, tem um caráter matemático, além de que os desafios enfrentados pelos alunos podem ser comparados àqueles enfrentados em outras disciplinas que envolvem abstração e raciocínio, necessários para a aprendizagem de muitos conceitos matemáticos.

Entretanto, a utilização de computadores abre novas perspectivas para a própria Matemática, ao vinculá-la a outras ciências, para as quais o processamento e o tratamento de dados, bem como os cálculos, são fundamentais para a resolução de vários problemas.

Segundo Friedmann (2005), a informatização de diversos setores da sociedade coloca

o conteúdo discreto da Matemática em evidência, levantando uma discussão para a inclusão dos mesmos no Ensino Fundamental. Problemas que envolvam a matemática discreta devem ser incluídos no ensino, de forma que o aluno exercite a forma algorítmica de pensar, unindo abstração, técnicas matemáticas e o fator temporal, a fim de viabilizar futuras soluções computacionais para os problemas.

É difícil separar as contribuições oriundas da Lógica e da Matemática para o desenvolvimento do pensamento algorítmico, que foram fundamentais para o *status* atual dos algoritmos e computadores.

Friedmann (2005, p. 20) salienta que,

> Resgatar parte das contribuições da Matemática à Ciência da Computação é um processo interessante sob o ponto de vista do ensino, pois possibilita a valorização de alguns conhecimentos matemáticos e como eles evoluíram a ponto de ganharem vida própria em outra ciência.
> Esse processo serve também para apontar aquelas deficiências que dificultam o entendimento em fazer conexões entre conceitos matemáticos e como eles são usados no estudo dos algoritmos, pois existem semelhanças e diferenças ao se transladar esses conceitos para a Algorítmica.
> O fato dessas deficiências no ensino da Matemática não serem abordadas, é um obstáculo para o entendimento do aluno sobre o que é pensar de forma algorítmica e sobre o que é um algoritmo. Esses questionamentos são relevantes em um mundo informatizado, pois estão relacionados com a resolução de problemas e com as limitações impostas pela realidade física dos computadores, os quais utilizam procedimentos algorítmicos para executar tarefas, sejam elas simples ou complexas.

Questões ligadas ao ensino e à aprendizagem têm sido objeto de estudo para pesquisadores das mais diversas áreas, porém na área de Ciências Exatas, ganharam destaque nos últimos anos, e muitos trabalhos têm sido realizados na tentativa de minimizar as deficiências encontradas.

O que costuma ocorrer é um ciclo vicioso. Os engenheiros que optam pelo magistério normalmente reproduzem a metodologia de ensino tradicional a que foram submetidos. No melhor caso, alguns procuram informalmente melhorar seu desempenho. São poucos os que têm uma formação didático-pedagógica sólida. Segundo Correia & Cheng (2001, p. 199),

> nas áreas de engenharia os professores são, geralmente, profissionais de reconhecida competência técnica sem que apresentem formação docente que os insira no ambiente pedagógico ativo e possibilite a reflexão sobre seu trabalho, de modo a contribuir com análises críticas também dos aspectos humano, social e político na produção de novas tecnologias.

Em vista da discussão anterior, este trabalho visa procurar subsídios para:

- Identificar e analisar algumas das principais dificuldades na aprendizagem de programação, mediante uma investigação das raízes da dificuldade em transpor procedimentos formulados de forma matemática (contínua) para sua correspondente solução algorítmica (discreta), que possa ser executada por um computador.

Para tanto, assumiremos as seguintes hipóteses:

- A discretização do raciocínio, ou seja, a elaboração de um raciocínio de forma discreta para um processo de natureza contínua, é um obstáculo epistemológico à aprendizagem.
- A identificação e manipulação da estrutura de repetição, que estão diretamente relacionadas a essa discretização, são obstáculos epistemológicos para a aprendizagem de algoritmos.
- A passagem de um tipo de registro de representação semiótica, a saber, a linguagem natural, para outro, a linguagem algorítmica, comporta um obstáculo à aprendizagem de algoritmos.

CAPÍTULO 2

O Algoritmo – Conceitualização e Evolução Histórica

A primeira vez que um algoritmo foi escrito para um computador foi em 1842, como mencionado no capítulo anterior, por Ada Byron, para a máquina desenvolvida por Charles Babbage, porém como Babbage não concluiu sua máquina, este algoritmo nunca foi implementado. Entretanto, a idéia de algoritmo habitava a consciência dos matemáticos desde o século XVII, especialmente em Leibniz. Todavia, foi na década de 1930 que essa idéia teve várias definições lógicas diferentes que originaram a moderna Ciência da Computação, entre elas, as que foram dadas por Gödel, Church, Post e Turing. Há mais de sessenta anos, Kurt Gödel, Alonzo Church, Emil Post e Alan Turing transformaram o conceito de algoritmo que era familiar e intuitivo, mas irremediavelmente obscuro, em um conceito tratável formalmente.

2.1 – O Advento do Algoritmo e suas Relações com a Matemática

Leibniz formulou há mais de dois séculos o projeto de criação de uma escrita universal, na qual todas as idéias compostas seriam expressas por meio de sinais convencionais para idéias simples, de acordo com regras fixas. Um exemplo aperfeiçoado no século XIX por Peano, é a definição de algo complicado, como a adição, em termos de algo mais simples como a sucessão. Neste caso, um “objeto” mental foi definido em termos de um “objeto” mecânico. Pode-se dizer, ainda, que algo infinito foi definido em termos de algo finito, enfim, algo mais complexo foi definido em termos de algo mais simples.

Regras elaboradas pela mente humana e baseadas em sinais e símbolos foram retratadas na forma de um algoritmo. Porém, apenas no século XX é que o conceito de algoritmo foi levado totalmente à consciência.

A especificação de um procedimento de decisão num sistema formal é um exemplo claro de algoritmo; o cálculo sentencial é um exemplo de sistema formal, e, dessa forma, o método das tabelas verdade é um exemplo de algoritmo. Muitos sistemas matemáticos são de natureza mecânica, embora, como qualquer máquina, impregnados com a inteligência de seu criador.

O primeiro conceito formal que correspondeu à noção de calculabilidade efetiva foi

dado por Gödel em 1931. Neste ano Gödel desenvolveu a teoria das funções recursivas, a classe de funções usada por ele, foi a das funções primitivas recursivas, definidas anteriormente por Dedekind em 1888 (DEDEKIND, 1969). Gödel (1990) definiu assim a recursividade: “As funções recursivas primitivas são precisamente aquelas funções aritméticas que podem ser derivadas do núcleo da recursividade por meio de um número finito de operações mecânicas específicas” (ISRAEL, 2002, p. 185). Isso tornou possível ao matemático falar sobre objetos infinitos a partir de regras finitas de construção.

Em 1936, com Alonzo Church, as funções recursivas que Gödel introduziu nos seus estudos sobre os teoremas da incompletude dão vez ao conceito de algoritmo. O cálculo λ estava à frente das idéias de seu tempo, e realmente encarnou em várias linguagens funcionais de computador, considerando que seu reticulado de iterações não era mais difícil para um computador do que qualquer outra operação mecânica (SOBRINHO, 1987).

A *máquina de Turing*, concebida pelo matemático Alan Turing, introduziu a idéia de uma máquina imaginária, o que tornou possível posteriormente projetar e construir os primeiros computadores digitais. Esta máquina é tanto um projeto arquitetônico quanto um planejamento de procedimentos. A arquitetura descreve as quatro partes da máquina, que são comuns a todas as máquinas de Turing, sendo composta por uma fita, um conjunto de símbolos, uma cabeça de leitura e um conjunto finito de estados. Os procedimentos compreendem as instruções, e embora sejam escritas no mesmo código e tenham o mesmo formato, variam de máquina para máquina (SAGASTUME, 2003).

A razão é despida de seu mistério por meio de uma série de passos mecânicos. Revelou-se que um universo totalmente abstrato está sob o controle de uma operação mecânica (BERLINSKI, 2002). Porém, cabe aqui ressaltar que muitos processos são essencialmente humanos e, assim, acessíveis somente à mente humana.

Gödel introduziu as funções recursivas no discurso lógico e Church, o mecanismo do cálculo *lambda*. Essas eram abstrações matemáticas, suas conexões com o conceito de algoritmo foram marcadas por uma cadeia longa e complexa de definições.

Não se pode afirmar se a máquina de Turing tornou possível, ou provocou o aparecimento da tecnologia, ou se ela teria surgido de qualquer forma. O que se sabe é que a máquina de Turing representou um papel muito importante na história do pensamento, fornecendo um modelo matemático simples e inteligente da velha idéia de algoritmo.

Emil Post, contemporâneo de Turing, antecipou vários fatos e descobertas com relação ao pensamento computacional. Seu trabalho resultou no projeto de uma máquina, com as mesmas idéias de Turing, a diferença entre elas é que Post projetou um "trabalhador" no lugar da cabeça de leitura da máquina de Turing. Sua máquina era totalmente simbólica; ele antecipou não tanto o computador, mas, sim, o *software*. Tanto Turing quanto Post criaram máquinas com o intuito de representar o mundo do pensamento, ou parte dele.

A máquina de Turing é um modelo matemático abstrato, formado por um "alfabeto" finito de símbolos, por uma fita de comprimento ilimitado, dividida em pequenas unidades sucessivas (células ou casas) e um mecanismo com um número finito de "estados". Esse mecanismo é capaz de ler, escrever ou apagar um símbolo em uma célula dada, podendo substituí-lo por outro, não colocar nada no seu lugar ou não alterar nada. A máquina também pode permanecer em posição imóvel ou deslocar a fita uma casa para frente ou para trás. As calculadoras de Pascal, de Babbage e o Colossus, assim como os computadores modernos, são concretizações da máquina de Turing.

As funções recursivas de Gödel eram precisamente as funções que podiam ser realizadas pela conversão *lambda* de Church, e as operações realizadas por essas funções eram precisamente as que podiam ser executadas por uma máquina de Turing ou por uma máquina de Post.

A chamada *tese de Church* não é um teorema susceptível de demonstração matemática, porém tem um estatuto de crença, cuja veracidade é baseada em evidências. Ela afirma que o que pode ser feito efetivamente, pode ser feito por uma máquina de Turing. A tese de de Church, também chamada de Church-Turing, pode ser vista como uma tentativa para a delimitação da extensão e dos limites da computação abstrata.

Para Hao Wang (1974), baseado em um comentário proferido por Gödel, a tese de Church foi uma das grandes conquistas da lógica desde os anos trinta, construindo uma definição absoluta de processo mecânico, também chamado de procedimento efetivo ou algoritmo, evidenciando seu caráter epistemológico. Wang afirma, ainda, que este foi o único conceito epistemológico básico relacionado com à matemática que fomos capazes de iluminar até agora. Gödel faz menção ao conceito de computabilidade dado pela tese de Church, porque o mesmo não depende de nenhum formalismo, considerando que várias abordagens, já citadas, tentaram caracterizar este conceito e obtiveram resultados que podem ser

considerados equivalentes.

A tese de Church se destacou, porque se baseou na análise filosófica do processo de computação humana, ou seja, tratou de analisar o mecanismo de raciocínio utilizado pelos seres humanos, levando em consideração o fato da experiência humana ser informal, bem como a experiência matemática ser precisa.

Essencialmente, a diferença mais aparente entre o mundo do computador, pelo menos na atualidade, e o mundo real, em que supomos que o pensamento humano discorre, é o caráter discreto do primeiro, contra o caráter contínuo do segundo. Uma situação análoga ocorre na relação entre o pensamento matemático e o pensamento computacional.

O computador reside em um mundo no qual o tempo é representado pelos números naturais inteiros, assim como os algoritmos. Pode-se dizer que um computador é um dispositivo, e uma máquina de Turing, um artefato intelectual. Para compreender o que um algoritmo está fazendo, é necessário compreender como é o processo discreto que está por trás dessa ação.

Sobre essas diferenças, destaca-se o trabalho de Friedmann (2005), *Algumas Questões sobre Algoritmos, Modelagem e Ensino*, resultado parcial de sua tese de doutorado. Em seu trabalho ela argumenta que:

> Os métodos matemáticos para resolução de problemas exigem rigor matemático no sentido de convergência para uma solução, mas não há, sob o ponto de vista estritamente matemático, uma preocupação com o tempo de execução do algoritmo. Essa preocupação é um dos enfoques e uma das contribuições da Algorítmica[2] para o tratamento de problemas e sua resolução; a questão do tempo tem uma importância significativa para o agente computacional que vai resolver o problema, no caso o computador, que na sua concepção é uma máquina abstrata, mas que operacionalmente está sujeito a trabalhar dentro de um intervalo de tempo finito.
> Além da questão do tempo no tratamento de diversos problemas, surge com a Algorítmica e com a informática, uma maior valorização da Matemática Discreta. (FRIEDMANN, 2005, p. 2)

Ela alerta para a necessidade de preparar alunos e professores para lidar com problemas que envolvem questões algorítmicas, não mais como um apêndice da modelagem do problema, mas como uma parte fundamental da modelagem, ressaltando as mudanças na maneira de encarar os problemas, de forma a aliar o fator tempo ao método matemático de resolução.

Observa-se que uma forma algorítmica de pensar associada à resolução de problemas pressupõe sempre uma solução, ou seja, para qualquer entrada deve haver pelo menos uma

[2] Friedmann entende a Algorítmica como sendo o campo do conhecimento que se dedica ao estudo sistemático e analítico de algoritmos.

saída, que possa ser considerada como uma solução para o problema. Para Friedmann (2005), existe um monitoramento do processo envolvido na forma algorítmica de pensar. Entretanto, no caso da resolução de problemas, para que uma saída seja considerada uma solução, ela deve estar adequada ao contexto em que a situação está inserida. A maneira algorítmica de pensar relacionada à resolução de problemas (teóricos ou práticos) é contextualizada, sendo que o contexto é um tipo de controle (monitoramento) sobre a situação.

Na matemática escolar, o discreto e o contínuo são associados aos processos básicos de contar e medir e não a formas de pensar, o que pode ser um obstáculo epistemológico às novas formas de pensar. Segundo Brolezzi (1996), não existe uma distinção cognitiva entre contar e medir e a relação entre ambas requer um estudo mais aprofundado. A medida tem por fundamento a idéia de comparação e subsequente ordem; aparentemente o processo de contar é mais complexo que o de medir (do ponto de vista cognitivo), o que pode ser um indício de que o pensamento algorítmico, que é de caráter discreto, teria uma lógica diferente e mais complexa que a do pensamento matemático contínuo.

Analisando o exemplo dado por Brolezzi (1996), considere as sequências de pontos seguintes,

1. • • • • •

2. • • • • • • • • •

Observamos que não é necessário contar o número de pontos para saber que a segunda sequência tem mais pontos que a primeira. Efetuamos um raciocínio que mede "automaticamente" o comprimento das duas sequências e deduz o resultado.

A maneira como a Matemática tem sido ensinada no Ensino Fundamental e Médio, sem dúvida traz consequências para o aprendizado de algoritmos no Ensino Superior, porém a sua análise foge do escopo deste trabalho.

2.2 – Considerações Sobre a Tese de Church-Turing

Retomando as observações apresentadas na seção anterior e fazendo uma incursão pela história da Matemática, cabe ressaltar que Hilbert, fundador da Escola Formalista, acreditava que "todo problema matemático bem definido deve ser necessariamente possível de exata solução, quer na forma de alguma resposta concreta à pergunta formulada, quer pela prova da impossibilidade de qualquer solução, e, com isto, o necessário fracasso de todas as tentativas

feitas por resolvê-lo” (SOBRINHO, 1987).

Em 1928, Hilbert e Ackermann diferenciaram o cálculo dos predicados de primeira ordem do de ordem superior. No Congresso de Bologna, Hilbert apresentou quatro problemas em aberto, sendo o último deles o problema da completude do cálculo de predicados de primeira ordem. Em 1930 e 1931, respectivamente, este último problema foi resolvido por Gödel afirmativamente, enquanto que negativamente para o cálculo de ordem superior. Os trabalhos de Gödel, que se seguiram, tiveram grande repercussão e representaram o limiar de uma nova era na Lógica Matemática.

Outro problema interessante, segundo Sobrinho (1987), dentro da temática hilbertiana era o problema da decisão. Parecia claro, para Hilbert, que com a solução desse problema seria possível, em princípio, solucionar todas as questões matemáticas de forma puramente mecânica. Esse problema pode ser traduzido para o caso dos algoritmos, como a necessidade de decidir se o algoritmo irá terminar em um tempo finito.

Analisando o seguinte trecho de um algoritmo, pode-se notar que ele iria ser executado infinitamente, pois a condição de parada nunca será realizada.

1. x = VERDADEIRO
2. **enquanto** x = VERDADEIRO
3. escreva (“Hoje está sol!”)
4. **fim_enquanto**.

O próximo passo rumo a novas concepções se deu nos trabalhos de Alonzo Church, no Instituto de Estudos Avançados em Princeton. Church trabalhava na solução do problema da Indecidibilidade com Kleene, sua abordagem era feita por meio das funções λ-definíveis, introduzidas por ele e Kleene. As funções λ-definíveis foram estudadas no λ-cálculo, que é o sistema precursor da linguagem de programação LISP.

Em 1933, Gödel visitou o Instituto, proferindo seminários, assistidos por Kleene. Nessa ocasião, Gödel introduziu a classe das funções recursivas gerais, e em conversa com Church, levantou a hipótese da identidade entre a classe das funções computáveis por procedimentos mecânicos, e das recursivas gerais.

Posteriormente, Church publicou um importante artigo no *The American Journal of Mathematics*, em que demonstrou a identidade entre a classe das funções recursivas gerais e a das funções λ-definíveis, e enunciou sua famosa tese, chamada por Kleene de Tese de Church,

mediante a qual a classe das funções computáveis, por procedimentos mecânicos, coincide com a classe das funções λ-definíveis.

Com a tese de Church, o conceito vago e intuitivo de “procedimento mecânico” ganha o *status* de entidade matemática precisamente definida, e, como aplicação, Church demonstrou que o cálculo de predicados é indecidível, resolvendo assim o *Problema da Indecidibilidade* em artigo publicado em 1936 no primeiro número do *The Journal of Symbolic Logic*, sob o título *A note on the Entscheidungsproblem*(CHURCH, 1936)).

Gödel afirma que o conceito de computabilidade dado pela tese de Church é independente de um formalismo particular, pois diversas abordagens foram testadas para caracterizar esse conceito, e todas obtiveram resultados equivalentes. Além das funções λ-definíveis de Church-Kleene ou das recursivas gerais de Gödel-Herbrand, cabe destacar as definições equivalentes de Alan Turing (1936), Emil Post (1943), S. C. Kleene (1952) e, finalmente, J. C. Shepherdson e H. E. Sturgis (1963).

De todas essas definições, a que teve maior relevância para a teoria da computação foi, sem dúvida, a do matemático inglês Alan Turing.

Turing teve contato com o problema da indecidibilidade durante um curso sobre os fundamentos da matemática, ministrado pelo topólogo Max Newman. Naquela ocasião, este problema foi citado como uma das principais questões em aberto nos fundamentos da matemática, e a expressão “procedimento mecânico” perturbou demasiadamente o jovem Turing no ano de 1935.

Durante aquele ano, Turing trabalhou no problema e produziu sua obra mais importante, intitulada *On computable numbers with an application to the Entscheidungsproblem*, em que analisa o ato de computar do “computador humano”. Como resultado, forneceu argumentos mediante os quais todas as computações podem ser efetuadas por suas máquinas, chamadas mais tarde de máquinas de Turing. Demonstrou a inexistência de uma solução positiva para o problema da decidibilidade.

Turing submeteu seu manuscrito à apreciação de Newman em meados de 1936; porém, em paralelo a seus trabalhos, nos Estados Unidos, Alonzo Church terminou seu artigo para publicação. Nesse artigo, Church antecipava vários resultados de Turing. Mesmo assim, com a influência de Newman o artigo de Turing foi publicado. Neste artigo estava formulada a tese de Church-Turing, que identifica as funções computáveis às funções λ-definíveis, ou às

Turing-computáveis, ou às funções recursivas gerais de Gödel- Herbrand.

A definição de *Máquina de Turing* é resultado de uma tentativa direta para formular matematicamente a noção de calculabilidade efetiva, enquanto que diversas outras noções foram originadas de maneiras distintas, e posteriormente associadas às funções efetivamente calculáveis. Segundo Wang (1974), o que Turing fez foi analisar o ato humano de calcular e, assim, chegar a um número de operações simples que são de natureza "obviamente" mecânica.

Kleene (KLEENE, 1976), observa que "todos os exemplos de funções efetivamente calculáveis e todas as operações utilizadas para definir tais funções a partir de outras cuja questão já tenha sido investigada, deram origem a funções recursivas gerais".

Segundo Turing (1936), são dois os princípios que regem os processos de computação, tornando-os mecânicos:

1. O princípio de determinação: o que o computador vai executar tem que ser predeterminado, ou seja, o comportamento do computador em cada momento fica determinado pelos símbolos que está observando, e pelo seu "estado mental" naquele momento.
2. O princípio da finitude: o número de estados mentais a serem considerados deve ser finito, assim como o número de símbolos.

O termo "estado mental" é utilizado para representar a capacidade que o ser humano tem de, a cada estágio de um dado raciocínio, escolher qual o passo a ser dado em seguida, pois o número de itens que a mente é capaz de armazenar é finito a cada momento.

A análise de Turing nos leva impreterivelmente à sua conclusão, que pode ser resumida como: a computabilidade efetiva é sinônino de calculabilidade por meio de suas máquinas.

Gödel trabalhou muito tempo com questões filosóficas relativas ao contraste entre mentes e máquinas. Como conclusão, chegou aos seguintes resultados:

Para ele a mente humana é incapaz de mecanizar todas as intuições matemáticas, fato este que poderia ser chamado de "incompletabilidade da Matemática", intimamente relacionado com o problema da incompletude. Outro resultado afirma que, ou a mente humana consegue ultrapassar qualquer máquina ou, então, existem questões da teoria dos números indecidíveis para a mente humana. Entretanto, Gödel, assim como Hilbert, se

recusava a admitir o comportamento “absolutamente irracional”, de acordo com o qual a mente seria capaz de formular questões que, pela sua própria natureza, lhe seja impossível responder.

A computação atual é baseada basicamente na idéia de Turing e posterior adaptação implementada por Von Neumann, o que certamente implica limitações. Todas as linguagens de programação, de uso geral, precisam ser “traduzidas” para a correspondente representação por meio de uma máquina de Turing. Isso significa que qualquer problema resolvido, utilizando uma determinada linguagem, deve poder ser resolvido em uma máquina de Turing. Ainda cabe ressaltar que, para que a linguagem possa ser considerada completa, ela deve solucionar todos os problemas que a máquina de Turing soluciona.

As limitações que a máquina de Turing apresenta são bem conhecidas e supostamente controladas. Podemos dizer que as limitações da computação são, na verdade, limitações do próprio sistema axiomático e sem mudanças significativas no paradigma da Matemática, é impossível conceber uma máquina de computar mais poderosa que o computador digital.

As linguagens de programação são apenas outras “línguas” que nos permitem conversar com computadores na forma de mnemônicos, sem ter que usá-los diretamente. No entanto, tudo aquilo que quisermos fazer está atrelado aos mnemônicos, e assim sendo quanto mais distante a linguagem estiver de uma máquina de Turing original, mais trabalhoso é o processo de tradução entre ambas (ERDTMANN, 2007).

2.3 – O Conceito de Algoritmo

Podemos dizer que resolver um problema computacional, ou melhor, uma classe de problemas, significa encontrar uma máquina de Turing, ou uma função recursiva, ou um mecanismo utilizando cálculo *lambda*, ou finalmente um algoritmo que solucione o problema em questão.

Neste trabalho, utilizaremos a definição operacional do conceito de algoritmo introduzida por Donald Knuth (1968). Para ele, um algoritmo é um conjunto de regras que, para um conjunto de entradas, irá produzir uma saída específica. Ou seja, para cada sistema de dados (entradas), ele realiza o processamento e, depois de um número finito de passos, chega a um resultado, podendo ser traduzido para uma linguagem de programação e executado por uma máquina.

O algoritmo de Euclides[3] é um bom exemplo de algoritmo. Segundo Knuth (1968), nos anos 1950, a palavra algoritmo era praticamente sinônimo de "algoritmo de Euclides". Podemos descrever o algoritmo de Euclides da seguinte forma:

Dado dois números inteiros positivos a e b, e considerando que $a \geq b$, se o resto da divisão de *a* por *b* for igual *a* zero, o MDC é o divisor (*b*). Caso contrário, realize uma nova divisão entre o divisor (*b*, que passa a ser o dividendo) e o resto (que passa a ser o divisor). Se o resto dessa nova divisão for igual a zero, o MDC é igual a esse novo divisor. Caso contrário, realize uma nova divisão entre o novo dividendo e o novo divisor, se o resto for igual a zero, o MDC é igual a esse novo divisor e assim sucessivamente. Este processo está exemplificado abaixo:

Exemplo 1: MDC (90, 36)

Passo 1: 90 div 36 ➔ quociente = 2; resto = 18

Passo 2: 36 div 18 ➔ quociente = 2; resto = 0 ➔ MDC = 18

Exemplo 2: MDC (81, 64)

Passo 1: 81 div 64 ➔ quociente = 1; resto = 17

Passo 2: 64 div 17 ➔ quociente = 3; resto = 13

Passo 3: 17 div 13 ➔ quociente = 1; resto = 4

Passo 4: 13 div 4 ➔ quociente = 3; resto = 1

Passo 5: 4 div 1 ➔ quociente = 4; resto = 0 ➔ MDC = 1

Para Knuth (1968, p.2), este procedimento de cálculo pode ser sumarizado da seguinte forma:

> "Dados dois números inteiros positivos *a* e *b*, encontre o máximo divisor comum (ou seja, o maior inteiro positivo que divide *a* e *b*).
> passo 1: [Cálculo do resto] Divida *a* por *b*. Seja *r* o resto;
> passo 2: [Verifique se o resto é nulo] Se *r* = 0, o algoritmo termina e o valor do MDC é *b*;
> passo 3: [Troca] Faça *a* = *b*, *b* = resto e volte para o passo 1. "

O trecho de algoritmo a seguir implementa o cálculo do MDC de Euclides em pseudocódigo, utilizando o processo iterativo.

Exemplo 1 - Algoritmo de Euclides utilizando o Processo Iterativo

Função mdcDeEuclides (dividendo: **inteiro**, divisor: **inteiro**): inteiro

[3] O algoritmo de Euclides busca encontrar o MDC (Máximo Divisor Comum) entre dois números inteiros diferentes de zero. É um dos algoritmos mais antigos conhecidos, desde que apareceu na obra *Elementos* de Euclides por volta de 300 a.C. (MILIES, 2003).

```
inicio
  inteiro c;
 enquanto resto (dividendo, divisor) ≠ 0  //calcula o resto da divisão de dividendo por divisor
          c ← resto (dividendo, divisor);
          dividendo ← divisor;
          divisor ← c;
  fim-enquanto
  retornar divisor
fim-funcao.
```

Entretanto, há uma maneira mais concisa para calcular o MDC, utilizando o processo Recursivo, como mostra o exemplo 2.

Exemplo 2 - Algoritmo de Euclides utilizando o Processo Recursivo

```
Função mdcDeEuclides (dividendo: inteiro, divisor: inteiro): inteiro
// calcula o mdc pelo método de Euclides usando recursividade
inicio
        se divisor = 0 entao            //Se r = 0, o algoritmo termina e o valor do MDC é b
                mdcDeEuclides ← dividendo
        senao mdcDeEuclides ← mdcDeEuclides (divisor, resto (dividendo, divisor))
        fim-se
//a operação resto (dividendo, divisor), resulta no resto da divisão de dividendo por divisor
fim-funcao.
```

Observa-se que o algoritmo recursivo está mais próximo do algoritmo matemático, ele é praticamente a própria definição do MDC, ao passo que para transformar este mesmo algoritmo na versão iterativa, é necessário discretizar os processos de repetição e condicionais que estão implícitos no modelo matemático.

Resumindo, as idéias de Knuth (1968), para que um método, um roteiro, ou um procedimento seja considerado um algoritmo, ele deve possuir clareza, precisão, um número finito de instruções e deve ter a sua execução terminada para quaisquer valores de dados.

CAPÍTULO 3

O Ensino de Algoritmos e sua Relação com a Educação Matemática

A seguir, são apresentados alguns trabalhos que deram subsídios para a nossa pesquisa: o primeiro aborda o ensino de algoritmos em cursos de computação, o segundo analisa os obstáculos epistemológicos no desenvolvimento do pensamento algébrico e o último traz uma proposta para a inclusão de problemas discretos no currículo de Matemática, nível fundamental e médio, com o objetivo de desenvolver o raciocínio lógico computacional.

O primeiro trabalho foi desenvolvido pela professora Lisbete Madsen Barbosa (BARBOSA, 2001), como dissertação de mestrado na área de Ensino de Matemática, na PUC de São Paulo. O que motivou a realização do trabalho foi a observação das dificuldades experimentadas pelos estudantes na aprendizagem da lógica de programação, representada por meio de algoritmos.

Assim como nós e muitos outros pesquisadores, a autora observou que apesar de muitos alunos conseguirem descrever como resolver um problema, apresentavam dificuldade em conceber a solução computacional (algorítmica).

O objetivo do trabalho foi investigar dois pontos principais:

1. Analisar a produção de algoritmos em linguagem natural, feita por estudantes de cursos iniciais, em algumas situações didáticas.
2. Comparar essas produções com as representações que podem ser feitas em pseudocódigo, para os mesmos algoritmos descritos pelos estudantes.

Concordamos com Barbosa (2001) que o processo de ensino-aprendizagem de desenvolvimento de algoritmos traz questões similares às do ensino-aprendizagem de Matemática, porque, em sua essência, ambos tratam de resolução de problemas e utilizam representações simbólicas peculiares.

Uma das hipóteses levantadas pela autora citada é que

> as dificuldades encontradas pelos estudantes na aprendizagem de algoritmos podem ter origem em obstáculos não apenas relacionados à conceituação. Fenômenos ligados à percepção, observação, representação semiótica, estrutura da mente podem prejudicar ou interromper a aprendizagem. (BARBOSA, 2001, p.17)

Considera-se que a elaboração de um algoritmo compreende duas etapas: a concepção e a representação. Como a descrição do algoritmo é feita em uma linguagem que não é

natural, acredita-se que o seu desenvolvimento mobilize dois registros de representação semiótica que, segundo Duval (1995), são necessários na formação dos conceitos. Para analisar as representações feitas pelos estudantes em linguagem natural e compará-las com as representações feitas em pseudocódigo, a pesquisadora elaborou uma sequência didática cujo foco principal foi o processo de desenvolvimento do algoritmo.

Os experimentos foram realizados com alunos iniciantes no estudo de algoritmos e pelos resultados obtidos concluiu-se que, normalmente, não houve congruência entre as duas representações. As situações didáticas foram baseadas principalmente nos trabalhos de Brousseau (1986) e Artigüe(1988).

Segundo Barbosa (2001, p. 18), "a criação e a representação de um algoritmo não podem ser consideradas como fases consecutivas de uma ação, uma vez que o algoritmo se põe a descoberto a partir de tratamento de sua representação".

Considerando que um algoritmo pode ser representado de diferentes formas, como exposto na seção 4.5 – Formas de Representação de um Algoritmo e diante do fato de muitos alunos afirmarem que sabem resolver o problema informalmente, mas não conseguem conceber a solução algorítmica, Barbosa (2001) observa que este pode ser um indício de que a dificuldade pode estar relacionada com a representação. Logo, a representação de um algoritmo pode constituir um obstáculo a sua elaboração.

O sistema de representação de algoritmos mais utilizado no ensino atualmente é o pseudocódigo. Esta preferência pode ser vista na maioria dos livros didáticos e técnicos sobre o assunto. Já a linguagem natural não aparece em nenhuma etapa desse processo.

Barbosa (2001) sentiu a necessidade de fazer a representação dos algoritmos em linguagem natural, com o objetivo de recuperar decisões feitas durante o desenvolvimento do algoritmo, possibilitando reflexões, generalizações e outras particularizações. Porém, as dificuldades com o pseudocódigo persistiram, o que levou a pesquisadora à seguinte suposição: se de posse de uma estratégia para a resolução de um problema, um estudante consegue fazer uma representação do algoritmo em linguagem natural, mas tem dificuldades em fazer a representação em pseudocódigo, um obstáculo a essa descrição pode ser a conversão entre registros de representação semiótica diferentes.

É interessante observar que estudos na área de Educação Matemática mostram que a passagem de uma representação para outra, muitas vezes não é evidente para a maioria dos

alunos, o que é apontado por Duval:

> A compreensão conceitual, a diferenciação e o domínio das diferentes formas de raciocínio, as interpretações hermenêutica e heurística dos enunciados estão intimamente ligados à mobilização e à articulação quase imediata de muitos registros de representação semiótica. A conversão de representações depende dessa coordenação. (DUVAL, 1995, p. 7)

As representações semióticas em Matemática são indispensáveis à comunicação e também são necessárias ao desenvolvimento da própria atividade matemática. Da mesma forma que na Matemática os algoritmos fazem referência a objetos que não são diretamente acessados pelos nossos sentidos ou por instrumentos. O acesso ao conhecimento de um tipo abstrato de dados só pode acontecer com o auxílio de alguma representação, da mesma forma como acontece, por exemplo, com o objeto ‘função’, na Matemática.

Por conta dessas semelhanças, a autora considera que as dificuldades encontradas no ensino-aprendizagem de algoritmos podem estar relacionadas com a questão da representação, sendo o ponto central da sua investigação a conversão do registro em linguagem natural para o registro em pseudocódigo.

O trabalho de pesquisa se restringiu ao aspecto da utilização da linguagem natural, intermediária para a descrição de algoritmos em pseudocódigo, e supôs que os estudantes não estudaram desenvolvimento de algoritmos anteriormente.

Os procedimentos de pesquisa seguiram os princípios da engenharia didática, conforme (ARTIGUE, 1988), envolvendo aproximadamente cento e dez alunos, separados em grupos de até cinco participantes. As atividades propostas para a realização dos experimentos podem ser caracterizadas como situações adidáticas, na forma de situações-problema.

A situação-problema traz um contrato didático diferente do convencional. Segundo (BROUSSEAU, 1986), o contrato didático estabelece convenções que devem gerenciar os comportamentos do professor e dos alunos. Deve-se ressaltar que o campo de saber em jogo nas situações propostas é o dos problemas computacionais. A organização da sequência didática, bem como de uma sequência de situações-problema, torna necessárias escolhas didáticas que devem ir ao encontro das etapas – ação, formulação e validação. As atividades preparadas foram propostas de forma que os alunos tivessem a oportunidade de criar procedimentos que representassem a noção de algoritmo.

Como conclusões foram encontradas diferenças significativas entre a linguagem natural

utilizada pelos alunos e seu correspondente pseudocódigo, principalmente nas estruturas de repetição. Dos vinte e um grupos que concluíram as atividades relativas à estrutura de repetição, apenas doze apresentaram alguma forma intuitiva de repetição, porém nenhum se preocupou em estabelecer o número de vezes que o processo deveria ser repetido.

Os resultados verificados evidenciam uma distância considerável entre as duas representações. Sugeriu-se um estudo mais aprofundado da conversão entre a linguagem natural e o pseudocódigo, considerado importante para o processo ensino-aprendizagem de algoritmos.

O segundo trabalho foi desenvolvido pela professora Elsa Malisani (MALISANI, 1999). Segundo Malisani, é na fase de transição entre o pensamento aritmético e o pensamento algébrico em que se encontra a passagem de um campo semiótico significativo, a saber, a aritmética, e a tentativa de introduzir uma nova linguagem, a álgebra, relativa a uma certa classe de problemas, a resolução de equações. Os obstáculos epistemológicos estão relacionados com esta passagem. Se considerarmos os algoritmos computacionais, observamos dificuldades em transpor procedimentos matemáticos aprendidos no Ensino Fundamental e Médio em algoritmos formais que solucionem computacionalmente o problema, o que também envolve uma mudança de representação.

Malisani (1999) apresentou um resumo dos erros que os alunos têm realizado ao resolver equações e problemas algébricos e os conceitos necessários na fase de transição entre o pensamento aritmético e o pensamento algébrico. Malisani (1999) entende a noção de erro de acordo com Brousseau (1986), em que o erro não é efeito da ignorância, mas uma consequência de um conhecimento anterior que se torna falso ou não apropriado a uma nova situação. Pode-se relacionar esta noção, à noção de obstáculo epistemológico, descrita por Bachelard (1938).

O uso do simbolismo adequado favorece o desenvolvimento do pensamento algébrico, por este motivo na história da álgebra tem importância não apenas a história dos conceitos, mas também o sistema de símbolos utilizados para expressá-los (ARZARELLO *et al*., 1994). A respeito do desenvolvimento da álgebra, é possível determinar três períodos históricos distintos:

1. Fase retórica: anterior a Diofanto de Alexandria (250 d.C.). Utilizavam apenas linguagem natural.

2. Fase sincopada: desde Diofanto até o final do séc. XVI. Surgiram algumas abreviaturas, mas os cálculos se desenvolvem em linguagem natural.
3. Fase simbólica: introduzida por Viète (1540 – 1603). Utilizavam letras e símbolos; a linguagem simbólica era utilizada para resolver equações e demonstrar regras gerais.

Fazendo um paralelo com a elaboração de algoritmos, podemos dizer que a descrição do algoritmo em linguagem natural corresponde à fase retórica; a formulação do algoritmo utilizando pseudocódigo corresponde à fase sincopada; e a implementação em uma linguagem de programação à fase simbólica.

O trabalho referido tem como objetivo estudar a construção da linguagem algébrica com sua ambiguidade semântica e sua riqueza de significados, em relação à evolução dos métodos e das estratégias de resolução de equações nos dois períodos históricos que precedem a formalização: retórico e sincopado, porque neles se desenvolveram precisamente as mudanças conceituais necessárias na fase de transição entre o pensamento aritmético e o pensamento algébrico.

Entre 1500 e 1600 foram introduzidos quase todos os símbolos conhecidos na atualidade. Foi um processo lento, pois a álgebra simbólica não suplantou de um só golpe a álgebra sincopada. Com Viète se produziu a “tradução” mais significativa na construção da linguagem simbólica.

Considerando os métodos de resolução, a análise histórica dos diversos procedimentos utilizados para resolver equações mostra a necessidade de recorrer sempre a vários tipos de linguagem: natural, aritmética e geométrica.

Para Malisani (1999), a semântica da linguagem algébrica, em relação à linguagem natural, é menos rica que a aritmética ou geométrica. Na fase sincopada é necessário apoiar-se nelas para formular regras, interpretar problemas, obter sua solução e para justificar as passagens algébricas. São a ambiguidade semântica e a riqueza de significados que permitem a utilização pouco a pouco da linguagem simbólica.

Por exemplo, na introdução dos números complexos, o obstáculo na compreensão desses entes como números, não dependia do tipo de equação ou do problema, senão do procedimento efetuado para a extração de raízes. A impossibilidade de efetuar um processo computacional fez sentir a necessidade de introduzir objetos algébricos de natureza mais abstrata: os números complexos.

Um aspecto relevante na construção da linguagem algébrica é a possibilidade de expressar simbolicamente a generalização dos problemas. Bombelli (1966) tenta generalizar os problemas resolvendo o problema aritmético de forma analítica. Primeiramente, formulava uma regra geral, além dos valores numéricos, por último, aplicava esta regra à resolução de uma equação análoga. Isso demonstra a importância que a linguagem algébrica assume nos processos de simbolização.

As principais conclusões desse trabalho foram as seguintes:

1. O desenvolvimento da linguagem algébrica levou ao abandono progressivo da linguagem natural como meio de expressão das noções algébricas.
2. Na fase de transição entre o pensamento aritmético e o pensamento algébrico, certos obstáculos no nível aritmético podem retardar o desenvolvimento da linguagem algébrica, por exemplo, a introdução dos número negativos, ao passo que a introdução de novas estratégias e de novos conteúdos algébricos pode encobrir conhecimentos aritméticos anteriores.
3. A necessidade de introduzir novos objetos, mais abstratos, aparece na impossibilidade de completar um procedimento de resolução de um problema particular, havendo necessidade de um processo computacional é o caso da introdução dos chamados números imaginários.
4. No processo de construção da linguagem algébrica é possível distinguir dois níveis para conceber a generalidade de um método: o primeiro é relativo à possibilidade de aplicá-lo a uma série de casos específicos e o segundo, refere-se à possibilidade de expressá-lo por meio da álgebra simbólica.

Novamente, remetendo à nossa questão, na elaboração de um algoritmo deve-se solucionar um problema específico e então buscar a generalização para a classe desse problema, por meio de uma forma de representação adequada.

O último trabalho apresentado nesta seção é a tese de doutorado, elaborada pela professora Clícia Friedmann, intitulada Matemática Discreta, Algoritmos, Modelos. Tendências do Ensino de Matemática no Início do Século XXI, na Universidade Federal do Rio de Janeiro (FRIEDMANN, 2005). A questão central é a discussão sobre a inclusão de assuntos relacionados à Matemática Discreta e algoritmos, por meio da modelagem Matemática, no currículo de matemática, nível fundamental e médio.

A autora escolheu o tema algoritmos, diante da sua relevância para diversas propostas curriculares, pois o uso crescente do computador permite abordagens distintas para problemas ligados à Matemática e a outras ciências. Entretanto, o tratamento dado a esses problemas exige novas formas de pensar, modelar e resolver situações utilizadas no dia a dia. A discussão elaborada por Friedmann (2005) é bastante relevante aos propósitos deste trabalho.

Concordamos que os conceitos matemáticos, ao serem transpostos para a computação, adquirem significados diferentes. A pesquisadora observou algumas dificuldades em alunos e professores para realizar a passagem de conceitos matemáticos para o algoritmo e vice-versa.

A finalidade do estudo sobre algoritmos apresentado, foi estabelecer comparações entre a forma tradicional com que alguns conceitos e estruturas matemáticas são estudados na escola (Ensino Fundamental e Médio) e a maneira de utilizá-los na estruturação e descrição de algoritmos.

Segundo Friedmann (2005), o ser humano, para lidar com computadores, tem que "limitar" sua forma de pensar, deve excluir a subjetividade, a ambiguidade e não pode omitir informações que clarifiquem as instruções a serem seguidas pela máquina. Isso implica a estruturação do pensamento em um número finito de passos bem delineados para que sejam seguidos por uma máquina. Essa tarefa, ainda que pareça quase mecânica, é na verdade uma forma matemática de pensar e um exercício difícil para o homem, devido a sua característica de se autoprogramar para resolver problemas. Essa dificuldade deve ser levada em consideração no ensino de Matemática relacionado a algoritmos e resolução de problemas.

A autora faz algumas conexões entre a Matemática Discreta, o ensino de Matemática e a resolução de problemas. Baseada nestes três aspectos propõe a utilização da modelagem matemática como forma de abordagem, pois ela privilegia a formação ativa de conteúdos. Um dos objetivos de Friedmann (2005) foi relacionar essa abordagem com a Matemática Discreta e algoritmos, considerando que os problemas escolhidos foram modelados dentro desse campo da Matemática e os métodos que conduzem para uma melhor solução para eles não podem ser implementados em tempo real. Assim sendo, não é possível garantir que as soluções encontradas sejam as mais eficientes.

Esses problemas são conhecidos como problemas de conjuntos dominantes (PCD). Segundo Friedmann (2005), esses tipos de problemas, além de estarem ligados à Matemática discreta, são um estímulo para a construção de diferentes modelos, e para a criação e a

implementação de técnicas algorítmicas. Na ausência de uma solução ótima possível, a criatividade dos alunos pode ser explorada, pois o interesse está no desenvolvimento de formas de pensar algorítmicas.

A tese não tem a intenção de contrapor a matemática discreta e a matemática contínua, pois diversos campos da Matemática têm relação com ambas, como a Álgebra e a Lógica. Para Maurer (1997), não existe uma definição exata para o que seja matemática discreta, o que proporciona seu florescimento em muitas direções.

Entre as definições que nos chamaram a atenção destacamos (FRIEDMANN, 2005, p. 12):

- "Matemática Finita: que abrange as situações descritas por conjuntos finitos".
- "Qualquer tipo de Matemática que possa ser feita em um número finito de passos".

Entre as conclusões da referida tese, ressaltamos aquelas que têm maior relevância para o nosso trabalho (FRIEDMANN, 2005):

- Embora exista estreita relação entre algoritmos e Matemática, é errôneo pensar que o aluno faça naturalmente uma ligação entre a matemática aprendida na escola e tópicos relacionados a algoritmos.
- Está surgindo um novo desafio para o professor, que é o de conciliar o ensino em contextos abstratos sugeridos pela Matemática com a realidade física dos computadores. Até o presente momento não há abstração, nem subjetividade no computador como instrumento de cálculo.
- Atualmente, quando se fala em Ciência da Computação em ambientes de ensino, principalmente nas universidades, existe o objetivo de ensinar o aluno a programar. Deveria existir concomitantemente um trabalho que desenvolvesse ou estimulasse "formas algorítmicas de pensar", associadas à resolução de problemas, as quais pressupõem sempre soluções para eles.
- Abordagens que utilizam modelagem e modelos em ambientes de ensino facilitam o cumprimento dos objetivos de o estudante adquirir a capacidade de argumentar e resolver problemas e de tornar-se capaz de transladar da realidade matemática para outras realidades e vice-versa.
- Problemas como os PCDs permitem o trabalho com modelagem e evidenciam a

necessidade de encontrar soluções de uma forma não exata, ou seja, utilizando procedimentos heurísticos. Possibilitam também mostrar algumas das diferenças e semelhanças nas maneiras de resolver problemas que são sugeridas pela Matemática e pela Ciência da Computação, sendo que esta última incorporou e deu importância ao fator tempo computacional na resolução de problemas.

- Uma situação modelada como um PCD pode ser enquadrada como um problema de otimização e foge àquelas estudadas no Cálculo Diferencial e Integral. Em termos de ensino e aprendizagem, é importante exemplificar que existem problemas que são adequados para serem modelados por matemática discreta, enquanto que outros são melhores equacionados quando se trabalha com matemática contínua.

A análise cuidadosa destes trabalhos nos deu subsídios para aprimorar nossa pesquisa e a escolha dos mesmos para estarem aqui apresentados, deu-se em função da suas relações com as dificuldades apresentadas no processo de ensino-aprendizagem nas áreas de Educação Matemática, Matemática, bem como na Ciência da Computação.

Cabe destacar que a noção de obstáculo, incluindo os epistemológicos, tem sido utilizada em diversos trabalhos referentes ao ensino da Matemática. Entretanto, na área de Informática não foi encontrado nenhum trabalho que estabeleça esta relação de forma explícita. Muitas pesquisas investigam as dificuldades na elaboração de algoritmos, sob óticas distintas, enfatizando a elaboração de ambientes informatizados, como veremos no capítulo 5, para o ensino dos mesmos, ou a especificação de uma linguagem de programação mais adequada para este fim.

Acreditamos que estes trabalhos têm seu valor, porém não contemplam as dificuldades por nós identificadas, uma vez que elas estão ligadas à própria concepção do algoritmo, e consequentemente ao desenvolvimento do raciocínio abstrato necessário ao pensamento computacional.

CAPÍTULO 4

Aspectos Didáticos e Epistemológicos na Aprendizagem de Algoritmos

Visando construir uma base teórica para este trabalho, pesquisamos assuntos ligados à Educação Matemática, bem como à Educação e Informática. Neste capítulo, apresentaremos os aspectos relevantes e que tiveram maior influência em nossa pesquisa.

O primeiro item abordado está relacionado à matemática discreta, considerando a estreita relação existente entre ela e os algoritmos; em seguida, discutimos o conceito de obstáculo epistemológico e a conexão com o processo de ensino e aprendizagem de algoritmos; por fim, aplicamos a teoria de registros de representação semiótica à representação de algoritmos e analisamos sua importância para o ensino/aprendizagem dos mesmos.

4.1- Matemática Discreta e Algoritmos

Entre os aspectos que relacionam os algoritmos com a matemática discreta, destacamos o fato de eles serem tipos de procedimentos que possuem um número finito de passos, o que coloca em evidência, a sua natureza discreta.

Nosso interesse pela matemática discreta não está apenas no fato de ela trabalhar com soluções para problemas por meio de um número finito de passos, mas na forma de elaborar raciocínios que possam ser expressos em uma sequência finita de passos. Como o ser humano não está acostumado a “discretizar” suas ações, esta é a primeira barreira a ser vencida na aprendizagem de algoritmos.

Para Friedmann (2003), agentes computacionais, humanos ou não, efetuam cálculos de maneira discreta, mesmo para solucionar problemas relacionados à matemática contínua, ou seja, do ponto de vista computacional há necessidade de discretizar o que é teoricamente contínuo. Isso evidencia o sentido oposto do que geralmente se faz quando se trabalha com modelos, em que há uma tendência de utilizar modelos contínuos para explicar e resolver problemas que também poderiam ser estudados de maneira discreta.

A Matemática possui ferramentas que permitem trabalhar teoricamente com sequências de cálculos que nem sempre são possíveis na prática. Os casos que tratam de divisões

exemplificam esse fato. Por exemplo, o resultado da divisão de qualquer número real pela metade é sempre possível; o quociente é um número real. Isso pode ser feito teoricamente quantas vezes se queira, mas existe um momento em que simplesmente não é factível efetuar as divisões. Um ser humano para automaticamente os cálculos quando não tem condições de continuar dividindo; já um computador precisa ser instruído sobre a precisão do cálculo que deve executar.

Nenhum agente computacional conhecido é capaz de calcular todas as imagens de uma função contínua, por exemplo $f(x) = x^2$, em um determinado intervalo, pois a função escolhida tem imagem para qualquer número real. Existiria um número infinito de cálculos, o que obrigaria um ser humano ou uma máquina a trabalhar continuamente, por tempo indeterminado.

Os gráficos da função mencionada no parágrafo anterior, e de outras funções que aparecem na tela do computador, são feitos utilizando cálculos aproximados e os pontos da curva são associados a um conjunto finito de pontos da tela. Em resumo, quando se usam computadores para trabalhar com funções contínuas e métodos associados a elas, que admitem muitos resultados pertinentes à matemática contínua, há necessidade de associar alguma aproximação discreta a eles, pois a matemática do computador é dessa natureza.

Essa aproximação não se dá de forma natural. Para analisar as dificuldades inerentes a esse processo, voltamos à questão da abstração, da necessidade de alteração da forma de compreender um conhecimento, para que seja possível transformá-lo. Essas questões nos remetem ao conceito de obstáculo epistemológico, que será discutido a seguir.

4.2 – Questões Didáticas e Epistemológicas

Com base no desenvolvimento histórico do conceito de algoritmo, tanto no campo da Matemática, quanto no campo da Informática, e nos pressupostos deste trabalho, consideramos que nosso objeto de investigação está ligado à forma de pensar sobre a solução de um problema e não sobre o próprio problema. O que nos levou a buscar na Educação Matemática os conceitos de ‘obstáculo’ e ‘ruptura’, em particular aqueles de caráter epistemológico.

Observamos que, para elaborar um algoritmo que resolva um problema dado, o conhecimento matemático estabelecido irá sofrer uma “ruptura epistemológica”, pois a necessidade de discretização traduzida em formas de raciocínio envolvendo processos iterativos, por exemplo, assim o requer. Essa ruptura está ligada a certos obstáculos

epistemológicos, pois se trata de uma mudança na forma de compreender um conhecimento, considerando que nos conteúdos matemáticos trabalhados até o final do Ensino Médio, os procedimentos utilizados têm um caráter essencialmente contínuo, enquanto que as soluções computacionais têm um caráter discreto, em que o tempo tem papel fundamental.

Entre vários trabalhos que discutem obstáculos epistemológicos, escolhemos alguns, de acordo com sua relevância para nosso trabalho, que pudessem nortear esta pesquisa. Destacamos o artigo publicado por Artigüe (1990), que tem como objetivo estabelecer uma relação entre epistemologia e didática, considerando as necessidades que surgem nos processos de conhecimento, envolvidos na formação e no desenvolvimento dos conceitos matemáticos.

Segundo Artigüe (1990), a análise epistemológica é necessária para o professor, porque proporciona uma historicidade aos conceitos e às noções matemáticas, além de auxiliar no controle das representações epistemológicas no ensino. O termo ‘representação epistemológica’ define as concepções que se formam no indivíduo, devido a sua vivência matemática.

A análise epistemológica permite ao professor medir e avaliar as diferenças existentes entre o “saber sábio” e o “saber ensinado”, termos que foram introduzidos por Y. Chevallard (1991). A escola se preocupa em ver nos objetos de ensino cópias simplificadas, porém fiéis, dos objetos da ciência. Para Artigüe (1990), a análise epistemológica nos permite compreender o que governa a evolução do conhecimento científico e nos ajuda a verificar a distância que separa os dois sistemas.

A divisão do saber sábio em partes que possam ser ensinadas, a um público determinado, permite que este saber seja baseado em um grupo restrito de competências. Em suma, a análise epistemológica ajuda o professor, no sentido de limpar os objetos de ensino de representações epistemológicas inadequadas, instaladas nestes com o passar do tempo.

Artigüe (1990) ao estabelecer uma relação entre os aspectos epistemológicos e a teoria das situações didáticas, observa que o papel do professor é o de construir o conhecimento num meio constituído para este fim. Deve-se considerar que o ensino da Matemática é mais do que a transmissão de um conhecimento, é a transmissão de uma cultura.

A análise epistemológica propõe ao professor uma série de perguntas, entre elas: O que transpor para o ensino referente aos aspectos culturais? Como fazê-lo de forma a não

adulterar o sentido da cultura em questão? Em quais condições isto é viável? Em que caso as transposições dependem do público a ser ensinado? Quais são os inconvenientes das transposições atuais? Quais são seus efeitos?

Nas disciplinas de introdução à programação deve-se perguntar: Os alunos têm algum contato com o uso de tecnologias computacionais? Em caso afirmativo, como se deu esse contato? Os alunos têm alguma noção de programação? Em caso afirmativo, de que forma? Qual o ambiente utilizado?

Considerando a popularização dos recursos tecnológicos e da informática, muitos estudantes acreditam que dominam a tecnologia, o que pode ser favorável ou não para a aprendizagem, pois existe uma diferença significativa em atuar como usuário de ferramentas computacionais e como criador dessas ferramentas. Essa diferença pode causar um sentimento de frustração entre os estudantes. É interessante que o professor possa integrar as questões de cunho epistemológico à sua atividade.

Segundo Schubring (2002), o interesse da Didática da Matemática com os obstáculos epistemológicos surgiu em decorrência da mudança de concepção da didática como uma técnica ou prática para uma ciência experimental e independente, o que aconteceu nos anos 1960 com a disseminação da nova Psicologia Genética de Piaget. Desde então, os erros dos alunos, subsídios para uma análise epistemológica, passaram para o centro da reflexão teórica da didática e da sua prática experimental.

A noção de ‘obstáculo epistemológico’ foi introduzida pelo filósofo francês Gaston Bachelard em 1938. Segundo ele, é no próprio conhecimento que aparecem, por uma classe de necessidade funcional, as lentidões e os problemas.

Apesar deste cientista não incluir, explicitamente, o ensino da Matemática em sua obra, o conceito de obstáculo epistemológico foi adotado pela Didática da Matemática, na França, a partir de meados dos anos 1970, sendo aplicada também em outros países (SCHUBRING, 2002).

Segundo Bachelard (1996), a evolução de um conhecimento pré-científico para um nível de conhecimento científico passa, quase sempre, pela rejeição de conhecimentos anteriores e se defronta com um certo número de obstáculos. Assim, estes obstáculos não se constituem na falta de conhecimento, mas, pelo contrário, são conhecimentos antigos, cristalizados pelo tempo, que resistem à instalação de novas concepções e de novas formas de

pensamento, que ameaçam a estabilidade intelectual de quem detém esse conhecimento.

Com relação à aprendizagem de algoritmos, podemos comparar o conhecimento da matemática elementar, referente à educação básica e nível médio, ao conhecimento pré-científico, citado por Bachelard. Este resiste face às exigências de mudança que o pensamento computacional nos traz, como, por exemplo as ligadas à discretização.

A introdução da noção de obstáculo epistemológico na Didática da Matemática se deu em 1976, na conferência da CIEAEM *(Comission internacional pour l'étude et l'amélioration de l'enseignement des mathématiques)* em Louvain la Neuve, com a publicação de um artigo de Brousseau sobre didática da matemática, abordando tal noção. Nesse artigo, a definição de obstáculo epistemológico referia-se a conhecimentos anteriores mal-adaptados, ressaltando a importância da análise epistemológica para detectar as dificuldades que são realmente inevitáveis, porque constituem o desenvolvimento do conhecimento. Para Brousseau, assim como para Bachelard, um obstáculo é conhecimento e não a falta dele.

Segundo Brousseau (1983, p.170), "um obstáculo de origem epistemológica, é verdadeiramente constitutivo do conhecimento, é aquele do qual não se pode escapar e que se pode em princípio encontrar na história do conceito." Ele está ligado à resistência do saber mal-adaptado, e o vê como um meio de interpretar alguns erros recorrentes e não aleatórios, cometidos pelos estudantes, quando lhes são ensinados alguns tópicos da Matemática. É na análise histórica dessas resistências, e nos debates, que devemos buscar os elementos que permitam identificar os obstáculos, sendo então necessário relacionar este estudo histórico com o estudo didático.

Assim, deve-se buscar construir situações de ensino que permitam superá-los.

O segundo artigo sobre obstáculos epistemológicos no ensino da Matemática, foi publicado por Glaeser (1981), em que ele identifica um conjunto de obstáculos no desenvolvimento da noção de números negativos. Entretanto, este artigo foi bastante criticado por Brousseau em 1983, levantando uma série de perguntas não consideradas por Glaeser, que ele acreditava deveriam ter sido feitas para que as dificuldades encontradas pudessem ser classificadas como obstáculos epistemológicos.

Segundo Artigüe (1990), Glaeser utilizou os termos 'obstáculo', 'dificuldade', 'barreira' e 'sintoma' de maneira ingênua, por estar convencido de que era prematuro fechar estes conceitos em formulações muito rígidas. Para ele, um dos objetivos mais importantes da

Didática da Matemática é o de determinar os obstáculos que se opõem à compreensão e à aprendizagem desta ciência.

Spagnolo (1996), completa a definição de obstáculo epistemológico utilizando uma interpretação semiótica das linguagens matemáticas. Para ele,

> o obstáculo epistemológico está relacionado com a passagem entre o campo semântico[4] significativo em uma certa época histórica da comunidade matemática e a tentativa da mesma introduzir uma nova linguagem relativa a uma certa classe de problemas. Os objetos matemáticos dos campos semânticos anteriores, que poderiam servir para a construção sintática, nos fundamentos da nova linguagem, são os obstáculos epistemológicos. (p. 81)

Observando o desenvolvimento histórico da Matemática, pode-se considerar, como exemplo, a Aritmética como campo semântico anterior, e a Álgebra como a nova linguagem.

É também de interesse deste trabalho a análise feita por Pais (2002) de que

> a existência de obstáculos epistemológicos em matemática se revela, muito mais, na fase da produção de uma demonstração do que de seu registro por meio do texto de uma demonstração. Tal como acontece na etapa de criação da matemática, durante a experiência da aprendizagem escolar há também um processo correspondente a uma redescoberta do saber, de onde os obstáculos podem, analogamente intervir diretamente no fenômeno cognitivo. (p. 42)

Durante a aprendizagem, ao iniciar o contato com um conceito inovador, pode ocorrer uma revolução interna entre o equilíbrio aparente do velho conhecimento e o saber que se encontra em fase de elaboração. Isso interessa à didática, pois para aprendizagem escolar, por vezes, é preciso que haja rupturas com o saber cotidiano.

Brousseau (1983) classifica os obstáculos relacionados ao ensino/aprendizagem da seguinte forma:

1. Obstáculos de origem ontogenética são os que sobrevivem de fato às limitações (neurofisiológicas, entre outras) do sujeito, no momento de seu desenvolvimento: o desenvolvimento de conhecimentos apropriados a seus meios e a seus objetivos.
2. Obstáculos de origem didática são aqueles que ocorrem em função do projeto do sistema educacional.
3. Obstáculos de origem epistemológica são os que estão relacionados com o aspecto histórico do conceito.

Brousseau (1997, p. 99) enumera os seguintes passos, que considera necessários para identificar os obstáculos epistemológicos:

---

[4] Segundo Stephen Ullman, "a teoria dos campos semânticos fornece um método valioso para abordar um problema difícil mas de crucial importância: a influência da linguagem no pensamento. Um campo semântico não reflecte apenas as ideias, os valores e as perspectivas da sociedade contemporânea; cristaliza-as e perpetua-as também; transmite às gerações vindouras uma análise já elaborada da experiência por meio da qual será visto o mundo, até que a análise se torne tão palpavelmente inadequada e antiquada que todo o campo tenha que ser refeito" (*Semântica*, 4ª ed., Fundação Calouste Gulbenkian, Lisboa, 1977, p. 523).

- Encontrar os erros recorrentes e mostrar que eles se reagrupam em torno de concepções.
- Encontrar os obstáculos na história da Matemática.
- Confrontar os obstáculos históricos com os obstáculos da aprendizagem e estabelecer seu caráter epistemológico.

Em seguida, ele retoma as características enunciadas por Dourox (1983):

A. Um obstáculo será um conhecimento, uma concepção e não uma dificuldade ou falta de conhecimento.
B. Esse conhecimento produz respostas adaptadas em um contexto frequentemente dado.
C. Esses conhecimentos geram respostas falsas fora deste contexto. Porque uma resposta correta exige uma visão diferenciada.
D. O conhecimento anterior resiste às condições a que está sendo confrontado e ao estabelecimento de um conhecimento posterior. Não basta possuir um conhecimento novo para que o precedente desapareça (é isto que diferencia a superação do obstáculo das idéias de Piaget). É indispensável identificar e incorporar seus inconvenientes dentro de um novo saber.
E. Depois da tomada de consciência da sua inexatidão, o obstáculo se manifesta de uma maneira intempestiva e obstinada.

A noção de obstáculo está ligada à produção de erros, induzida pelo conhecimento anterior, ou pelas noções pré-científicas, como diria Bachelard, levando a respostas falsas. Por exemplo, quando pedimos aos alunos que elaborem um algoritmo computacional para implementar o método de Euclides para o cálculo do M.D.C. (descrito na seção 2.3), embora estes alunos saibam utilizar o referido método, da forma tradicional, muitas vezes não conseguem alterar esse conhecimento e expressá-lo de forma algorítmica. Entenda-se aqui forma algorítmica no sentido computacional. Ou seja, o conhecimento anterior do método de Euclides resiste às alterações necessárias para conceber o algoritmo computacional correspondente.

Para transformar o algoritmo matemático para o cálculo do M.D.C. no correspondente computacional, é necessário sistematizar os processos de repetição e os condicionais que estão implícitos no modelo matemático.

Artigüe levanta diversas questões sobre como assegurar que um dado conhecimento é ou não um obstáculo epistemológico, entre elas:

- Para que possa ser considerado um obstáculo é necessário comprovação histórica de dificuldades análogas?
- Uma transposição didática eficaz da noção de obstáculo epistemológico deve permanecer ligada à história?

Além da atribuição do papel de obstáculo epistemológico a uma dificuldade, devemos nos preocupar com os mecanismos produtores de obstáculos. Artigüe (1990, p. 13) identifica alguns destes mecanismos tanto diacronicamente, como sincronicamente. Entre eles: a regularização formal abusiva. Como exemplos de resultante desse processo de regularização, ela apresenta erros do tipo: $(a+b)^2 = a^2 + b^2$ ou $\sqrt{a+b} = \sqrt{a} + \sqrt{b}$; a fixação sobre uma contextualização ou uma modelização familiar; e o amálgama das noções sobre um suporte dado. Também são fontes de obstáculos os processos de generalização, dentre outros.

Segundo Pais (2002), o interesse em estudar a noção de obstáculo decorre do fato da mesma permitir identificar as fontes de diversos fatores que levam a aprendizagem a uma situação de inércia e de obstrução. Baseado nas idéias de Bachelard destaca que é preciso entender como ocorre a reorganização intelectual de modo que o novo conhecimento entre em harmonia com os anteriores, sendo esse o momento em que os obstáculos se manifestam.

Artigüe (1990) conclui que o que fundamenta uma classe de obstáculos epistemológicos é: primeiro sua aparição e resistência na história dos conceitos considerados, e então a observação de concepções análogas nos estudantes, mais que a constância da resistência a estas concepções para os estudantes atuais. Talvez o caráter histórico dos obstáculos, atribuído por Brousseau, tenha origem na essência evolutiva dos conceitos envolvidos.

Na história do desenvolvimento de um conceito, o professor pode encontrar os sinais de resistência à aprendizagem, porém não pode utilizar a investigação como prova da resistência a estes conceitos pelos estudantes atuais. Artigüe (1990) constata que os obstáculos de origem epistemológica são reforçados por outros obstáculos de outra origem, normalmente origem didática. E, ainda, levanta a possibilidade de, considerando as diferenças entre as condições de gênese histórica e escolar, haver a existência de “nós de resistência”, que

funcionam como os obstáculos epistemológicos no interior do desenvolvimento das matemáticas, sem que seja possível atribuir-lhes um papel histórico de obstáculo.

Entre outros trabalhos nessa direção, destaca-se a tese sobre a aprendizagem da noção de limite, sustentada por Cornu (1983) e os trabalhos de Sierpinska (1985), cujo objetivo é identificar os obstáculos na aprendizagem desta noção e auxiliar os alunos a superá-los. Estes autores conservam a noção de obstáculo, conforme Bachelard.

Além do trabalho de Cornu, outro trabalho mencionado por Artigüe (1990) trata dos obstáculos epistemológicos na didática da física, desenvolvido por L. Viennot (VIENNOT, 1977, *apud* ARTIGUE, 1990). Segundo Viennot, as investigações sobre didática da Física têm conduzido a uma oposição do conhecimento comum e o conhecimento científico. O conhecimento comum tem como características: estado de evidência, imprecisão e ambiguidade das formulações e falta de coerência interna, o que vai de encontro ao conhecimento científico.

Em suas investigações, Viennot encontra algumas regularidades na aprendizagem de certos conteúdos. Estas regularidades levam a uma tendência comportamental peculiar. Considerando que os obstáculos parecem estar ligados a grandes tendências de raciocínios, pergunta-se: Estas formas de raciocínio aparecem apenas nos principiantes ou nos mais experientes também?

O que classificava o aluno como experiente não era a superação do obstáculo, no sentido ingênuo, mas, sim, a possibilidade de avaliar diferentes registros e entre estes, considerando que alguns chegam a controlar o obstáculo, ter a capacidade de escolher o mais adequado.

Tendo em vista os diversos aspectos teóricos ligados ao conceito de obstáculo epistemológico investigados e o nosso interesse em abordar questões relacionadas ao ensino e à aprendizagem de algoritmos, entendemos a noção de obstáculo neste trabalho como sendo de três tipos:

1. Obstáculo didático, é aquele ligado ao sistema de ensino. Incluímos nesta categoria obstáculos ligados ao planejamento de ensino, metodologia utilizada, número de alunos e a conduta do professor.
2. Obstáculo cognitivo, é aquele ligado aos processos de aprendizagem, dizem respeito ao aluno e estão ligados às particularidades do sujeito.
3. Obstáculo epistemológico é aquele relacionado ao próprio conhecimento, ligado ao

conceito, neste caso, ao conceito de algoritmo, indo ao encontro da definição elaborada por Spagnolo (1996), citada acima.

4.3 – Registros de Representação Semiótica

Duval (2003) propõe uma abordagem cognitiva para avaliar as dificuldades dos alunos na compreensão da Matemática e a natureza dessas dificuldades. Neste trabalho estendemos esta abordagem, para avaliar as dificuldades na aprendizagem de Introdução à Programação. Segundo Duval (2003, p. 13),

> Comumente, em análises do que consiste a compreensão em matemática e na procura da razão dos bloqueios de compreensão que muitos alunos experimentam, evocam-se os conceitos matemáticos e suas complexidades epistemológicas, os quais podem ser explicados pela história de suas descobertas. Porém, uma tal abordagem não é suficiente para caracterizar aquilo que faz a originalidade e a especificidade do funcionamento do pensamento em matemática em relação a outros domínios do conhecimento científico. A diferença entre a atividade cognitiva requerida pela matemática e aquela requerida em outros domínios do conhecimento não deve ser procurada nos conceitos mas em duas características:
> 1. a importância das representações semióticas para o desenvolvimento da matemática e
> 2. a variedade de representações semióticas utilizadas em matemática.

Observa-se que este autor utiliza a palavra compreensão no sentido da aprendizagem, ligado aos aspectos cognitivos. Segundo ele, a representação semiótica se refere a um sistema particular de símbolos, por exemplo, a linguagem natural, a escrita algébrica, os gráficos cartesianos e a possibilidade de conversão em representações equivalentes em outro sistema semiótico, que podem, porém, assumir significados diferentes para o sujeito que as está utilizando. A noção de representação semiótica pressupõe a consideração de sistemas semióticos diferentes e uma operação cognitiva de conversão das representações de um sistema semiótico para outro. Para Duval (1995), essa operação pode ser descrita em primeiro lugar como uma troca na forma como um conhecimento é representado, por exemplo, a curva que representa uma equação de segundo grau e a expressão analítica em formato algébrico.

Registro de representação semiótica é o nome dado por Duval (1995) ao registro que se presta às operações cognitivas de formação, tratamento e conversão, chamado daqui por diante de registro de representação.

Duval (2003) afirma que o progresso do conhecimento está acompanhado da criação e do desenvolvimento de sistemas semióticos novos e específicos que coexistem com o

primeiro deles, a saber, o da linguagem natural. Segundo ele, existem quatro tipos de registro de representação, como mostra o Quadro 1.

	Representação Discursiva	**Representação Não-Discursiva**
Registros Multifuncionais: (Os Tratamentos não são algoritmizáveis)	Língua natural Associações verbais Forma de raciocinar: • argumentação • dedução	Figuras geométricas planas ou em perspectivas: • apreensão operatória e não somente perceptiva • construção com instrumentos
Registros Monofuncionais: (Os Tratamentos são principalmente algoritmos)	Sistemas de escritas: • numéricas (binária, decimal, fracionária,...) • algébricas • simbólicas (línguas formais) Cálculo	Gráficos cartesianos. • Mudanças de sistema de coordenadas • Interpolação, extrapolação

Quadro 1- Classificação dos diferentes registros mobilizáveis no funcionamento matemático. (DUVAL, 2003, p. 14)

Para Duval (2003, p. 14), "a originalidade da atividade matemática está na mobilização simultânea de ao menos dois registros de representação ao mesmo tempo, ou na possibilidade de trocar de registro de representação". Por exemplo, para representar uma função, podemos utilizar ora a representação gráfica, ora a expressão algébrica. A capacidade de escolher o tipo de registro e a utilização ora de um, ora de outro, é chamada de coordenação de registros. Observa-se que a análise deste autor sobre a "atividade matemática" se dá sob a perspectiva da aprendizagem.

Uma solução proposta para um problema pode privilegiar um registro específico, mas

deve haver possibilidade de representá-la utilizando outro registro. Segundo Duval (2003), para haver compreensão em Matemática é preciso coordenar ao menos dois registros de representação. Ele levanta a seguinte questão: “Essa capacidade de coordenação é adquirida de forma espontânea pelos alunos durante o ensino de matemática”? (DUVAL, 2003, p. 15).

Considerando o ensino de algoritmos, pergunta-se: Se os alunos, ao concluírem o Ensino Médio possuíssem essa capacidade, as dificuldades na aprendizagem de algoritmos seriam as mesmas?

Com o objetivo de responder a essa questão, formulamos uma atividade em que, na primeira fase, o aluno deveria elaborar uma solução matemática intuitiva, utilizando qualquer registro de representação, tanto a linguagem natural, quanto a algébrica e, na segunda fase, baseado na solução proposta na fase anterior, elaborar a solução computacional, ou seja, elaborar o algoritmo. Essa atividade faz parte de um conjunto de casos analisados neste trabalho, e está descrita detalhadamente na seção 7.2.

Existem dois tipos de transformação de representações semióticas: os tratamentos e as conversões. Para Duval (1995; 2003), o tratamento é a transformação de uma representação de partida em uma representação de chegada, dentro de um mesmo sistema semiótico. Para ele,

> A paráfrase e a inferência são as formas de tratamento em língua natural. O cálculo é uma forma de tratamento próprio às estruturas simbólicas (cálculo numérico, cálculo algébrico, cálculo posicional,...), a reconfiguração é um tipo de tratamento particular para as figuras geométricas, é uma das várias operações que dá ao registro das figuras seu papel heurístico. ... Há, naturalmente, regras de tratamento próprias a cada registro. Sua natureza e número variam consideravelmente de um registro a outro: regras de derivação, regras de coerência temática, regras associativas de continuidade e similitude ... No registro da língua natural, há paradoxalmente um elevado número de regras de conformidade e poucas regras de tratamento para a expressão discursiva de um enunciado completo. (DUVAL, 1995, p. 41)

A conversão acontece quando há mudança de sistema de representação, mas a referência aos objetos é conservada. Segundo Duval (1995, p. 40), “A conversão é a transformação da representação de um objeto, de uma situação ou de uma informação dada num determinado registro em outra representação desse mesmo objeto, dessa mesma situação ou a mesma informação num outro registro”.

Ela é uma transformação externa ao registro de representação de partida. Converter uma representação significa mudar a forma de representar um objeto do conhecimento, ou

seja, mudar a forma pela qual tal objeto está sendo ou foi representado.

No caso dos números racionais, por exemplo, a passagem de um registro de representação numérico-decimal para fracionário e percentual ou entre cada um deles, (0,1 = 1/10 = 10/100 = 10%), só é possível de se estabelecer quando o sujeito perceber que se trata de um registro de representação numérico, cuja diferença está na forma de apresentação da representação e não no objeto/conteúdo representado. Quando se faz a passagem do registro decimal para o fracionário ou percentual, temos uma conversão, e quando se estabelece a equivalência entre 1/10 e 10/100, o tratamento. Entretanto, para o aluno, tanto o tratamento quanto a conversão não são processos simples de se estabelecer.

A conversão enfrenta os fenômenos de não-congruência, pois os alunos não reconhecem o mesmo objeto por meio de duas representações diferentes, e os fatores de não-congruência podem mudar de acordo com os tipos de registro entre os quais a conversão é efetuada.

Deve-se tomar o devido cuidado para não confundir o tratamento com a atividade matemática de conversão, considerando que a conversão deve ser efetuada para permitir tratamentos diferentes em sistemas de registros de representação também diferentes. Por exemplo, quando trabalhamos com problemas, o mais importante é estabelecer as possíveis relações entre o enunciado, a representação intermediária e o tratamento matemático, uma vez que este objeto não é claro e acessível como os objetos físicos e, exatamente por isso, seu tratamento depende de uma representação semiótica.

Segundo Duval (1995), as representações semióticas não são necessárias apenas para fins de comunicação, elas desempenham papel fundamental para as atividades cognitivas do pensamento, não são meramente uma exteriorização das representações mentais, necessárias para se estabelecer uma comunicação, pois uma vez que o indivíduo que aprende, necessita delas para elaborar o conhecimento, portanto, desempenham as funções de cognição (tratamento, conversão e representação).

Para Duval (2003), a utilização de vários registros de representação propicia o desenvolvimento do conhecimento humano e possibilita a criação de novos sistemas semióticos, a exemplo da evolução nos sistemas de numeração utilizados pela humanidade no decorrer dos tempos. O progresso do conhecimento é oriundo da criação e

desenvolvimento de novos e específicos sistemas semióticos, resultado do trabalho com vários registros de representação. A criação de novos registros está diretamente relacionada às necessidades da espécie humana.

Da mesma forma que na Matemática, outras áreas de conhecimento também propiciam a criação de novos tipos de registro de representação. No caso da Informática, a teoria de Church-Turing, descrita no capítulo 2, possibilitou representar computacionalmente o pensamento humano, ou parte dele, por meio da Máquina de Turing. Sendo assim, esta máquina pode ser vista como a primeira forma de representação computacional.

Considerando o estado atual das tecnologias disponíveis para o funcionamento dos computadores, fez-se necessário a criação de um registro de representação computacional. Cabe aqui uma ressalva, Duval (1995) chama de representações internas ou computacionais aquelas que estão relacionadas ao tratamento e se caracterizam pela execução automática de uma determinada tarefa. É o registro mecânico que se faz de um determinado objeto. Elas são internas e não conscientes do sujeito. O sujeito apenas as executa, utilizando para isso regras, fórmulas ou esquemas, sem pensar em todos os passos necessários para sua execução.

De acordo com o autor (1993), as representações computacionais traduzem informações externas de um sistema, em uma forma que seja possível recuperá-las e combiná-las no interior do sistema.

Neste trabalho, diferente de Duval, iremos nos referir ao termo ‘registro computacional’, como o registro que pode ser “compreendido”, ou seja, executado por um computador, por meio de ferramentas disponíveis para tal (linguagens de programação, tradutores e compiladores em geral). Como já mencionado anteriormente, para não ficar dependente das tendências de mercado, com o surgimento de novas linguagens e técnicas de programação, a utilização do pseudocódigo (portugol), permite a generalização necessária para a aprendizagem de algoritmos.

4.3.1 – Registros de Representações Computacionais

Para conceber um algoritmo que resolva um problema, em primeiro lugar é necessário elaborar uma estratégia para solucionar o referido problema. De posse dessa estratégia, deve-se escolher a melhor forma de construir o algoritmo. Pode-se dividir este processo nas

seguintes etapas:

- Análise do enunciado do problema proposto, identificando os dados (variáveis) de entrada e saída.
- Definição das ações necessárias para que, baseados nos dados de entrada, seja possível obter os dados de saída desejados.
- A escolha de uma seqüência de passos para que estas ações sejam realizadas.
- Escrita do algoritmo na linguagem escolhida (fluxograma, pseudocódigo, linguagem C, etc.).

Entretanto, deve-se observar que, como o algoritmo representa a solução do problema, o fenômeno da não-congruência é um fato. Considerando que nesse processo, são mobilizados diferentes tipos de registros de representação, partindo de um registro multifuncional, a linguagem natural, na qual possivelmente foi elaborado o enunciado, passando para um registro monofuncional, que pode ser: algébrico, numérico ou simbólico. Nessa fase, pode haver a utilização de um registro misto, pois em alguns momentos a linguagem natural é utilizada em conjunto com a linguagem algébrica, simbólica e com o pseudocódigo. Finalmente, o algoritmo é escrito, utilizando um registro também monofuncional, o qual chamamos de *registro computacional*.

Novamente, é importante distinguirmos o que Duval (2003) chama de algoritmo, ao fazer a classificação dos registros monofuncionais. Para ele, registros monofuncionais são aqueles em que os tratamentos são principalmente algoritmos, sendo que ele utiliza o termo ‘algoritmo’ como os algoritmos conhecidos, utilizados pela Matemática, como o algoritmo de Euclides, descrito na seção 2.3. Estamos falando da concepção desses algoritmos, ou seja, uma vez que sabemos como funciona o algoritmo de Euclides, desejamos propor uma solução que possa ser implementada por meio de um computador, ou outra máquina compatível, como uma calculadora programável, para executar o tal algoritmo.

Segundo Duval (2003), para que a passagem de uma representação a outra aconteça de maneira espontânea, elas devem ser congruentes. Para que isso aconteça, é necessário: correspondência semântica entre as unidades significantes, igualdade na ordem de apreensão das unidades e as representações e conversão de uma unidade significante na representação de partida em uma só unidade significante na representação de chegada. Quando esses fatores não acontecem, as representações são ditas não-congruentes.

Observa-se que a coordenação entre as representações, que provém de sistemas semióticos diferentes, não é espontânea. A interpretação do enunciado de um problema, por exemplo, a formulação algorítmica de um problema matemático, necessária à sua compreensão e consequentemente solução, passa pelos mesmos problemas, já apontados no caso da Matemática. Segundo Duval (1995), a congruência ou a não-congruência são fatores determinantes para o sucesso ou fracasso, na solução de problemas que envolvem a troca do sistema de representação semiótica. Quanto maior for a não-congruência entre estes sistemas, mais complexas serão as atividades de tratamento, necessárias para realizar a conversão, podendo haver casos em que a mesma é impossível.

De acordo com Duval (2003), os três principais obstáculos na aprendizagem relativa a raciocínios, tratamento lógico-matemático e processos afins, ocorrem pelas seguintes razões:

1. Diversificação de registros de representação semiótica. Em particular, a diferença existente entre a linguagem natural e as linguagens simbólicas.
2. Diferenças entre o representante e o representado, ou entre a forma e o conteúdo de uma representação.
3. Coordenação entre diferentes tipos de registros de representação.

É importante escolher o registro de representação adequado, considerando que cada registro permite efetuar tratamentos diversos. A escolha correta pode facilitar ou não a aprendizagem.

Duval (2003) alerta para o fato da aprendizagem da Matemática se limitar, quase sempre, a um único tipo de registro, e, ainda, quando há mobilização de vários registros, simultânea ou sucessivamente, não há coordenação entre os mesmos, o que possibilita aos alunos apenas a compreensão parcial, mas não a ideal. Esta compreensão monorregistro se constitui em um obstáculo maior, quando há mudança no contexto no qual se realizou a aprendizagem, e a maioria dos alunos se mostra incapaz de mobilizar os conhecimentos adquiridos, que, teoricamente, todos deveriam “dominar” para solucionar a nova situação.

Duval (2003) afirma que, de um modo geral, a compreensão monorregistro é uma compreensão que não permite realizar transferências. Para que essas transferências possam ocorrer de modo natural, a aprendizagem deve abranger a coordenação dos registros de representação, ou seja, é preciso compreender que os diferentes registros referem-se ao mesmo objeto matemático e podem se complementar no sentido de que um registro pode

expressar características ou propriedades do objeto matemático que não são expressas com clareza em outro registro.

Se essa coordenação não tem sido possível no escopo das atividades que envolvem a Matemática, sem dúvida quando consideramos a elaboração de um algoritmo, a situação é ainda pior, pois a alteração de representação é claramente não-congruente.

Para efetuar uma conversão é necessário ter clareza quanto às unidades significantes no registro de partida e de chegada. Quando essa troca de registro envolve a natureza dessas unidades significantes, acontecem os maiores obstáculos. Por exemplo, as linguagens formais têm unidades discretas, enquanto as linguagens naturais apresentam vários níveis de determinação funcional das unidades.

Considerando o algoritmo de Euclides, observamos a passagem de um registro multifuncional (linguagem natural), para um registro misto incluindo linguagem numérica e natural e, finalmente, o algoritmo em linguagem computacional (pseudocódigo).

O processo de elaboração do algoritmo de Euclides será apresentado novamente, com o intuito de ilustrar o processo de conversão realizado. Esse algoritmo pode ser feito de duas formas, a primeira utilizando o processo iterativo; nesse caso, a conversão é mais difícil, devido à não-congruência entre os registros de representação. A segunda forma é utilizando o processo recursivo, em que a não-congruência é bem menos evidente, e facilita a atividade de conversão.

O processo de elaboração do algoritmo de Euclides será apresentado novamente, com o intuito de ilustrar o processo de conversão realizado.

1. Representação em linguagem natural:

Dados dois números inteiros *a* e *b*, e considerando que $a \geq b$, se o resto da divisão de *a* por *b* for igual a zero, o MDC é o divisor (*b*). Caso contrário, realize uma nova divisão entre o divisor (*b*, que passa a ser o dividendo) e o resto (que passa a ser o divisor). Se o resto dessa nova divisão for igual a zero, o MDC é igual a esse novo divisor. Caso contrário, realize uma nova divisão entre o novo dividendo e o novo divisor, se o resto for igual a zero, o MDC é igual a esse novo divisor e assim sucessivamente.

2. Representação em registro misto, linguagem natural e numérica:

MDC (90, 36)

Passo 1: 90 div 36 ➔ quociente = 2; resto = 18

Passo 2: 36 div 18 ➔ quociente = 2; resto = 0 MDC = 18

Quando Knuth (1968, p. 2), simplifica o enunciado do problema e sistematiza os passos para sua solução, já é possível verificar uma mudança da linguagem natural para uma representação mista:

> Dados dois números inteiros positivos *a* e *b*, encontre o máximo divisor comum (ou seja, o maior inteiro positivo que divide *a* e *b*).
> passo 1: [Cálculo do resto] Divida *a* por *b*. Seja *r* o resto;
> passo 2: [Verifique se o resto é nulo] Se *r* = 0, o algoritmo termina e o valor do MDC é *b*;
> passo 3: [Troca] Faça *a* = *b*, *b* = resto e volte para o passo 1.

3. Representação em registro computacional (pseudocódigo):

Apresentamos a seguir, duas versões de algoritmos computacionais para o algoritmo de Euclides, com o intuito de comparar o processo de conversão necessário a cada caso.

a) Algoritmo de Euclides, utilizando o Processo Iterativo

```
Função mdcDeEuclides (dividendo: inteiro, divisor: inteiro): inteiro
inicio
  inteiro c;
 enquanto resto (dividendo, divisor) ≠ 0  //calcula o resto da divisão de dividendo por divisor
        c ← resto (dividendo, divisor);
        dividendo ← divisor;
        divisor ← c;
  fim-enquanto
  retornar divisor
fim-funcao.
```

b) Algoritmo de Euclides utilizando o Processo Recursivo

```
Função mdcDeEuclides (dividendo: inteiro, divisor: inteiro): inteiro
// calcula o mdc pelo método de Euclides usando recursividade
inicio
      se divisor = 0 entao          //Se r = 0, o algoritmo termina e o valor do MDC é b
            mdcDeEuclides ← dividendo
      senao mdcDeEuclides ← mdcDeEuclides (divisor, resto (dividendo, divisor))
      fim-se
//a operação resto (dividendo, divisor), resulta no resto da divisão de dividendo por divisor
```

fim-funcao.

A chamada de uma função ocorre no algoritmo principal. O exemplo abaixo pode ser utilizado tanto para chamar a função iterativa quanto a recursiva.

```
// Algoritmo Principal, exemplo da chamada da função
Inicio
    inteiro a, b, mdc;
    escreva ("Entre com dois valores, para o cálculo do M.D.C.");
    leia (a, b);
    mdc ← mdcDeEuclides (a, b);
    escreva ("O m.d.c. é =", mdc);
fim.
```

O Quadro 2 apresenta a representação intermediária do Método de Euclides e a versão iterativa do algoritmo em pseudocódigo.

Algoritmo de Euclides Iterativo	**Representação Intermediária do Método de Euclides**
1. **inicio** 2. **inteiro** c; 3. **enquanto** resto (dividendo, divisor) ≠ 0 4. c ← resto (dividendo, divisor); 5. dividendo ← divisor; 6. divisor ← c; 7. **fim-enquanto** 8. **retornar** divisor **9. fim-funcao.**	passo 1: [Cálculo do resto] Divida *dividendo* por *divisor*. Seja *r* o resto; passo 2: [Verifique se o resto é nulo] Se *r* = 0, o algoritmo termina e o valor do MDC é *divisor*; passo 3: [Troca] Faça *a* = *b*, *b* = resto e volte para o passo 1.

Quadro 2 – Comparação entre as unidades significantes no registro de partida e de chegada, para a versão iterativa do algoritmo.

A estrutura de repetição, linha 3, será executada enquanto a condição for verdadeira, ou seja enquanto o resto da divisão entre o dividendo e divisor for diferente de zero; quando o resto da divisão entre o dividendo e divisor for igual a zero, a repetição é encerrada, e o

valor da variável *divisor* corresponde ao MDC procurado. Observando o Quadro 2, assumindo *a = dividendo* e *b = divisor*, vemos que os passos 1 e 2 são representados pela condição da estrutura de repetição, e pelo comando de atribuição, da linha 4, em que a variável *c* recebe o resto atualizado entre *dividendo* e *divisor*, já o passo 3 corresponde as linhas 5 e 6, dentro da estrutura de repetição.

Neste caso percebemos as unidades significantes no registro de partida e de chegada não são congruentes.

O Quadro 3 apresenta a representação intermediária do Método de Euclides e a versão recursiva do algoritmo.

Algoritmo de Euclides Recursivo	**Representação Intermediária do Método de Euclides**
1. **inicio** 2. **se** divisor = 0 **entao** 3. mdcDeEuclides ← dividendo 4. **senao** mdcDeEuclides ← 5. mdcDeEuclides(divisor, resto (dividendo, divisor)) 6. **fim-se** 7. **fim-funcao**.	passo 1: [Cálculo do resto] Divida *dividendo* por *divisor*. Seja *r* o resto; passo 2: [Verifique se o resto é nulo] Se *r* = 0, o algoritmo termina e o valor do MDC é *divisor*; passo 3: [Troca] Faça *a* = *b*, *b* = resto e volte para o passo 1.

Quadro 3 - Comparação entre as unidades significantes no registro de partida e de chegada, para a versão Recursiva.

A primeira vez que a função é chamada, *dividendo* = *a* e *divisor* = *b*, o que corresponde ao passo 1, considerando que o *divisor* seja > 0, a função se "auto" chama, o que caracteriza a recursão, só que dessa vez, com o *divisor* no lugar do *dividendo* e com o resto da divisão entre *dividendo* e *divisor* no lugar do *divisor*, o que corresponde ao passo 2, até que o resto entre *dividendo* e *divisor* seja igual a zero, o que corresponde ao passo 3. Então, as chamadas vão sendo substituídas pelos resultados intermediários até chegar ao mdc final.

Comparando o Quadro 2 com o Quadro 3, observamos que as unidades significantes no registro de partida, a saber, a descrição dos passos para o cálculo do mdc, e as unidades significantes no registro de chegada, respectivamente, os algoritmos iterativo e recursivo, estão mais próximas no algoritmo recursivo, pois esta versão é praticamente a sequência de

passos descrita por Knuth; ao passo que para transformar este mesmo algoritmo na versão iterativa, é necessário discretizar os processos de repetição e condicionais, que estão implícitos no modelo matemático.

Analisando a história do conceito de algoritmo, observamos que a utilização da teoria, elaborada por Turing, na programação de computadores, é dependente das tecnologias para a construção do *hardware*. Essa dependência se constitui um obstáculo, pois o formato dos algoritmos, para que os mesmos possam ser executados pela máquina, é determinado por esta tecnologia.

Não só nos computadores, mas também em todos os outros recursos tecnológicos disponíveis atualmente, a linguagem simbólica, utilizada pelas máquinas, é baseada em um sistema de numeração específico, como binário, ou hexadecimal. Essa limitação física faz com que tenhamos que "traduzir" nossos "pensamentos" de forma que o *hardware* possa "compreendê-los" e executá-los. Para que essa tradução seja possível, é necessário trabalhar com representações semióticas distintas, até chegarmos naquela utilizada pela máquina. Esse processo envolve não apenas formas de representação, como também a discretização de um processo de natureza contínua.

Neste processo, temos que considerar vários fatores produtores de obstáculos, os de origem epistemológica, as dificuldades relacionadas à conversão de registros de representação não-congruentes e os obstáculos de origem didática.

Para avaliar cada um destes, aspectos realizamos uma pesquisa empírica, descrita nos capítulos 6 e 7.

4.4 - A Aprendizagem de Algoritmos

Como já mencionado, para que o computador execute as tarefas que desejamos, devemos estabelecer qual a lógica de programação que ele deve utilizar. Considerando que o raciocínio é algo abstrato, intangível, os seres humanos têm a capacidade de expressá-lo por meio da palavra falada ou escrita, baseada em um determinado idioma, que segue uma série de padrões definidos por uma gramática. Um mesmo raciocínio pode ser expresso em qualquer um dos inúmeros idiomas existentes, mas continuará representando o mesmo raciocínio, usando apenas uma outra convenção.

O mesmo ocorre com a lógica de programação, que pode ser concebida pela mente

treinada e pode ser representada em qualquer uma das inúmeras linguagens de programação existentes. Estas, por sua vez, são atreladas a uma grande diversidade de detalhes computacionais, que pouco têm a ver com o raciocínio original. Para escapar dessa torre de Babel e, ao mesmo tempo, representar mais fielmente o raciocínio da lógica de programação, utilizamos o pseudocódigo.

Farrer (1999, p. 16), descreve um algoritmo da seguinte forma: “Ação é um acontecimento que, a partir de um estado inicial, após um período de tempo finito, produz um estado final previsível e bem definido. Portanto, um algoritmo é a descrição de um conjunto de comandos que, obedecidos, resultam numa sucessão finita de ações”.

Como vimos, podemos encontrar diversas definições para o termo algoritmo. Todas concordam que um algoritmo deve realizar um conjunto de tarefas para resolver um problema que esteja bem definido. Na medida que precisamos especificar uma sequência de passos, precisamos ordenar o pensamento; portanto, precisamos elaborar um raciocínio lógico (FORBELLONE, 2000).

Se pensarmos em atividades rotineiras que realizamos diariamente, podemos constatar que, muitas vezes, estamos intuitivamente executando um algoritmo, ainda que não-estruturado, por exemplo, quando utilizamos uma receita de bolo. Nela está descrita uma série de ingredientes necessários e uma sequência de passos (ações) que devem ser fielmente cumpridos para que se consiga fazer o bolo desejado.

Quando elaboramos um algoritmo, devemos especificar ações claras e precisas, que, a partir de um estado inicial, após um período de tempo finito, produzem um estado final previsível e bem definido. Isso significa que o algoritmo fixa um padrão de comportamento a ser seguido, uma norma de execução a ser trilhada, com vistas a alcançar, como resultado final, a solução de um problema, garantindo que, sempre que executado sob as mesmas condições, produza o mesmo resultado.

Um algoritmo tem por objetivo descrever o raciocínio lógico envolvido na solução de um problema. Dessa forma, viabiliza a abstração de uma série de detalhes computacionais, que podem ser acrescentados mais tarde, o que nos permite focalizar nossa atenção naquilo que é importante: a lógica de programação.

Um fator relevante na construção de algoritmos é que, uma vez concebida uma solução algorítmica para um problema, esta pode ser traduzida para qualquer linguagem de

programação e ser agregada às funcionalidades disponíveis nos diversos ambientes existentes.

Há muitas formas de se resolver um problema, pois cada pessoa pensa e age de maneira diferente. Cada indivíduo tem uma heurística própria, associada a padrões conhecidos por ele, e não a um objeto concreto. Isso significa que, para cada problema proposto, podem haver diversas estratégias para encontrar a solução. A prática e o bom senso indicarão qual é a mais adequada, que, com menor esforço e maior objetividade, produz o resultado almejado.

O ponto de partida para o aprendizado da lógica de programação, como muitos autores apontam, é quase que na sua totalidade a implementação de procedimentos matemáticos. Por exemplo, para o aluno construir um algoritmo computacional para encontrar o fatorial de um número *n*, ele deve utilizar o procedimento matemático intuitivo, aprendido no ensino fundamental, para encontrar o fatorial desse número.

Entretanto, a realidade abstrata da Matemática nem sempre corresponde à realidade física que cerca o ser humano, com leis que regem tempo e espaço. O homem é um agente computacional capaz de vivenciar as potencialidades e limitações desses dois mundos, o "real" e o "abstrato". Isso não ocorre com os computadores, que, apesar de serem fruto de uma concepção matemática abstrata, são feitos para resolver problemas no mundo real.

Os algoritmos podem ser tão complexos que os seres humanos precisam do auxílio de máquinas (computadores) para implementá-los; para que essas máquinas trabalhem de forma coerente, elas devem seguir regras rígidas, organizadas e formalizadas. A Matemática pode ser útil nesta organização, pois viabiliza a formalização adequada do conteúdo destes algoritmos.

4.5 – Formas de Representação de um Algoritmo

Um algoritmo pode ser representado de diversas maneiras, entre elas:

a) pseudocódigo, também chamado de pseudolinguagem ou portugol: é uma linguagem com alguma liberdade de sintaxe e parecida com a linguagem natural (português).
b) Fluxograma: é a representação em modo gráfico de um algoritmo, em que formas geométricas diferentes indicam as ações (comandos) a serem realizados.
c) Linguagem de programação: é um conjunto de regras sintáticas e semânticas usadas para definir um programa a ser executado por um computador.

Todas têm regras claras e bem definidas, embora apresentem diferenças quanto à sintaxe

dos comandos, possuem estruturas similares. Tanto o pseudocódigo, quanto o fluxograma, são representações intermediárias entre a linguagem natural e a linguagem de programação. A vantagem de utilizá-los está na abstração de detalhes de ambientes de programação, que não interferem na lógica do processo.

Apresentamos, a seguir, um exemplo de algoritmo expresso em cada uma das três formas de representação citadas. Este algoritmo calcula a média de duas notas, fornecidas pelo usuário, e se a média for maior que sete, dá uma mensagem de aprovado, caso contrário, dá uma mensagem de reprovado.

a) Algoritmo representado em portugol para o cálculo da média:

```
inicio          // início do algoritmo em portugol
   inteiro nota1, nota2;
   real media;
   escreva ("Entre com a primeira nota");
   leia (nota1);
   escreva ("Entre com a segunda nota");
   leia (nota2);
   media ← ( nota1 + nota2 ) / 2;
   se media > 7 entao
      escreva ("Aprovado");
   senao
      escreva ("Reprovado");
fim.
```

b) Algoritmo representado por fluxograma para o cálculo da média:

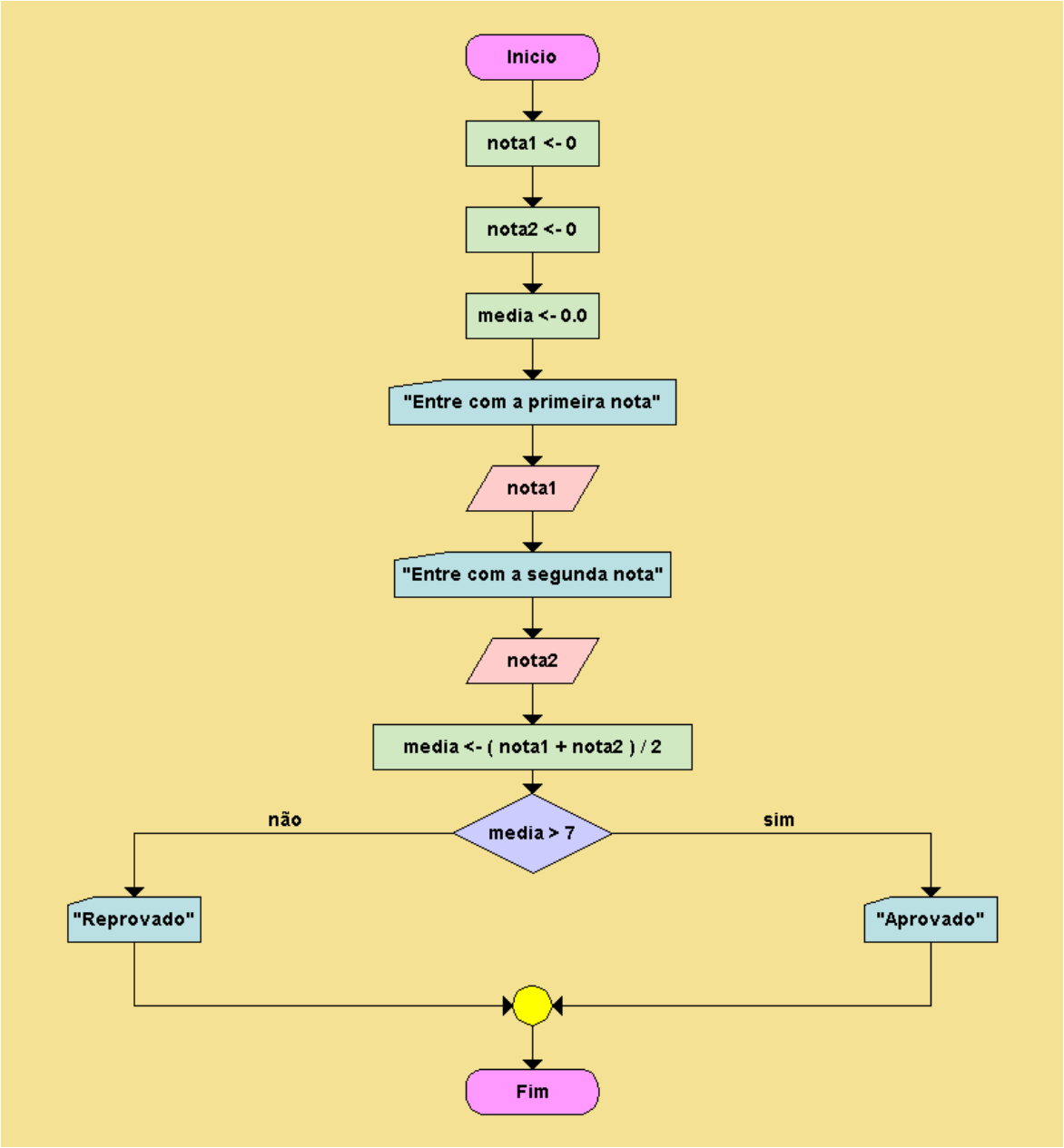


Figura 1 - Fluxograma do algoritmo para o cálculo da média

Fonte: A autora.

c) Algoritmo representado em Linguagem C, para o cálculo da média:

```
Programa
void main()
{
        int nota1, nota2;
        float media;
        clrscr();                       // Limpa a tela
        cin >> nota1 >> nota2;          // Recebe as duas notas
        media = (nota1 + nota2) / 2;    // Calcula a média
        if (media > 7)                  // Mostra o resultado
            cout << "\nAprovado";
        else
            cout << "\nReprovado" ;
}
```

4.6 - Estruturas para o Desenvolvimento de Algoritmos

Para especificar um algoritmo utilizando pseudocódigo, de maneira que seja possível executá-lo por meio de um computador, é necessário adotarmos uma linguagem com sintaxe rígida. Os elementos desse pseudocódigo, abordados neste trabalho, são:

- variáveis
- comandos de entrada e saída
- operadores aritméticos, relacionais e lógicos
- estruturas condicionais
- estruturas de repetição/laço

Na programação, uma variável pode assumir qualquer valor do seu conjunto domínio. As linguagens de programação contêm alguns conjuntos domínio pré-definidos, os quais são denominados de tipos de dados nativos, são eles: Inteiro, Real, Caracter e Lógico.

Quando um algoritmo depende de informações externas, que deverão de algum modo ser inseridas no computador, é necessário haver uma entrada de dados. Por exemplo, um programa que emite extrato bancário precisa que o usuário forneça o número da agência, a

conta corrente e a senha. Estes dados podem ser captados via cartão magnético ou teclado, e serão algumas das variáveis do programa.

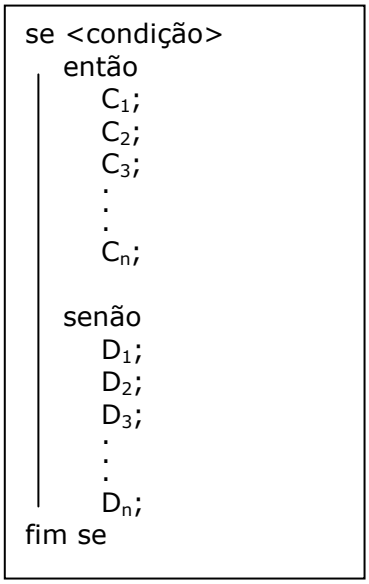
Uma vez que os dados de entrada tenham sido processados, mostram-se ou gravam-se os resultados num dispositivo de saída: impressora, monitor de vídeo, discos, etc. No caso do exemplo de extrato bancário, a saída poderia ser o saldo da conta corrente e os lançamentos efetuados.

Os operadores aritméticos e relacionais são os mesmos utilizados pela Matemática; já os operadores lógicos são os operadores elementares da Lógica Booleana *e* e *ou*.

4.6.1 - Estruturas Condicionais

A estrutura condicional permite a escolha do grupo de ações e estruturas a serem executados quando determinadas condições, representadas por expressões lógicas, são ou não satisfeitas. A estrutura condicional pode ser simples ou composta.

Quando a expressão condicional é verdadeira, o bloco de comandos que segue a palavra "então" é executado; caso contrário, é executado o bloco que segue a palavra senão. Observando o diagrama abaixo, se a condição for verdadeira, então a sequência de comandos C1;...; Cn é executada; caso contrário, se a condição for falsa, a sequência D1;...; Dn será executada.



4.6.2 - Estruturas de Repetição

As estruturas de repetição permitem que um bloco de comandos seja executado uma ou mais vezes, quando determinadas condições, representadas por expressões lógicas, são ou não satisfeitas.

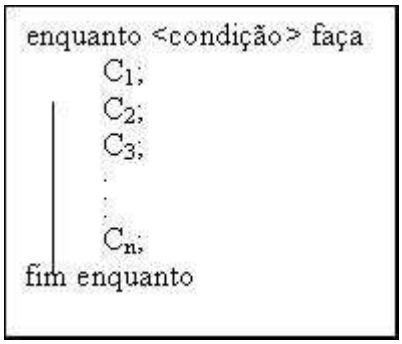
1. Repetição com teste no início, em que os comandos são executados apenas se a condição resultar em valor verdadeiro; neste caso, o comando utilizado chama-se Enquanto.

2. Repetição com teste no final, em que os comandos são executados pelo menos uma vez e, então, a condição é avaliada. Caso a condição resulte em valor verdadeiro, o processo é repetido. Este comando chama-se Repita - Enquanto.

3. Para: repete um bloco de comandos *k* vezes, em que *k* é uma quantidade determinada.

1. Enquanto (teste no início)

A sequência de comandos C1;...; Cn é executada enquanto a <condição> for verdadeira. Quando ela se torna falsa, a execução do algoritmo passa ao comando seguinte.



2. Repita – enquanto (teste no final)

A sequência de comandos C1;...; Cn é executada uma vez, então, se a <condição> for verdadeira, a execução se repete enquanto a <condição> continuar verdadeira. Quando ela se torna falsa, a execução do programa passa ao comando seguinte.

```
repita
    C1;
    C2;
    C3;
    .
    .
    Cn;
enquanto <condição> ;
```

3. Para

O comando inicia com a execução da atribuição <*v* ← i>, se a condição <v = ▷ for verdadeira, a sequência de comandos C1;...; Cn é executada, caso contrário, o comando é finalizado; em seguida, executa-se a atribuição <*v*← v+*p*> e todo o processo é repetido, exceto a atribuição.

```
para <v ← i> até <v = ▷ <v← v+p>
   C1;
   C2;
   C3;
   .
   .
   Cn;
fim para
```

Nos exemplos da utilização das estruturas de repetição mostrados a seguir, no Quadro 4, utilizamos as mesmas variáveis para manter a referência. Em todos os casos, a variável *A* vai tendo seu valor acrescido com o da variável *I*, que é incrementada em 1 a cada execução dos comandos de repetição. A coluna da esquerda mostra o trecho do algoritmo e as colunas da direita mostram o conteúdo das variáveis de acordo com a execução do algoritmo.

<table>
<tr><td>Repetição com teste no final
(repita enquanto)</td><td colspan="2">Acompanhamento da execução do algoritmo</td></tr>
<tr><td rowspan="6">inteiro A, I;
I ← 1; A ← 0;
repita
A ← A + I;
Escreva A;
I ← I + 1;
enquanto I< 5;</td><td>I</td><td>A</td></tr>
<tr><td>1</td><td>0</td></tr>
<tr><td>2</td><td>1</td></tr>
<tr><td>3</td><td>3</td></tr>
<tr><td>4</td><td>6</td></tr>
<tr><td>5</td><td>10</td></tr>
<tr><td>Repetição com teste no início
(Enquanto fim-enquanto)</td><td colspan="2">Acompanhamento da execução do algoritmo</td></tr>
<tr><td rowspan="6">inteiro A, I;
I ← 1; A ← 0;
enquanto I < 5 faça
A ← A + I;
escreva A;
I ← I + 1;
Fim-enquanto</td><td>I</td><td>A</td></tr>
<tr><td>1</td><td>0</td></tr>
<tr><td>2</td><td>1</td></tr>
<tr><td>3</td><td>3</td></tr>
<tr><td>4</td><td>6</td></tr>
<tr><td>5</td><td>10</td></tr>
<tr><td>Repetição com para</td><td colspan="2">Acompanhamento da execução do algoritmo</td></tr>
<tr><td rowspan="6">inteiro A, I;
A ← 0;
para I de 1 até 4 passo 1 faça
A ← A + I;
escreva A;
fim-para</td><td>I</td><td>A</td></tr>
<tr><td>1</td><td>0</td></tr>
<tr><td>2</td><td>1</td></tr>
<tr><td>3</td><td>3</td></tr>
<tr><td>4</td><td>6</td></tr>
<tr><td></td><td>10</td></tr>
</table>

Quadro 4 - Exemplos da utilização das estruturas de repetição

CAPÍTULO 5

Ambientes de Aprendizagem para Algoritmos

O ensino e a aprendizagem de introdução à programação tem sido objeto de estudo de diversos grupos de pesquisa tanto em universidades brasileiras quanto no exterior, entre elas podemos citar: (GUIMARÃES, 1995), (HUMEL, 2002), (HENDERSON, 1986), (JENKINS , 2001), COSTELLOE (2004a, 2004b). Estes estudos resultaram em metodologias e ferramentas utilizadas para promover a aprendizagem dessas disciplinas. Costelloe (2004a, 2004b), em seus trabalhos, discute e categoriza diversas delas. Um dos fatores relevantes, apontado por ela, é a necessidade de mais experiências empíricas para avaliar a eficácia dessas metodologias.

O que diferencia nossa pesquisa das pesquisas analisadas por Costelloe é que nosso enfoque está na concepção e no desenvolvimento do raciocínio computacional, enquanto que aquelas dão ênfase à tecnologia envolvida no tratamento dos problemas de ensino-aprendizagem de algoritmos, bem como às diversas técnicas didáticas utilizadas, mais do que aos problemas de caráter epistemológico.

O objetivo de apresentar este levantamento é procurar nessas abordagens subsídios para uma pesquisa teórico-prática sobre o problema que nos ocupa: teórica, na procura dos fundamentos epistemológicos do problema, e prática, na elaboração de um estudo de campo que avalie possibilidades, tanto tecnológicas como metodológicas, de melhoria das condições de ensino-aprendizagem de algoritmos.

Na construção de um programa, podemos identificar duas fases distintas: a resolução de problemas e sua posterior implementação. Segundo Henderson (1986), a resolução de problemas e o pensamento analítico são a maior fragilidade dos alunos nos cursos de Informática, para ele, esses tópicos deveriam ser enfatizados. Em seu artigo *Modern Introductory Computer Science* (HENDERSON, 1987) observa que um fator importante para a aprendizagem de resolução de problemas é aprender a usar a abstração; assim, os cursos deveriam oferecer aos alunos ferramentas para pensar de maneira abstrata, em que a fundamentação matemática deve ser fortalecida.

Na fase de implementação, o aluno deve escrever a solução encontrada para um problema, utilizando uma linguagem de programação, para, posteriormente, executar o

programa e corrigir seus possíveis problemas, com auxílio de um ambiente específico (como: Borland C., J. Builder, Pascal, entre outros)

Esta fase envolve uma série de áreas de conhecimentos que o estudante deve dominar, entre elas:

- Sintaxe e escolha da linguagem: muitos estudantes gastam suas energias tentando solucionar erros de sintaxe, ao invés de investir na elaboração da solução.
- Construtores de programa: mesmo comandos simples podem causar problemas aos alunos iniciantes. O que acontece, muitas vezes, é que os alunos compreendem os exemplos trabalhados em sala de aula, porém, ao tentarem resolver um novo problema, esbarram nas mesmas dificuldades, patinam e não conseguem sair do lugar.
- Ambiente de desenvolvimento: para editar, compilar e executar um programa é necessário a utilização de um ambiente de desenvolvimento integrado (IDES). Estes Ambientes não são adequados para programadores iniciantes, considerando que a maioria não possui suporte à resolução de problemas.
- Teste e Debug[5]: em alguns ambientes as mensagens de erro são complexas e confundem ainda mais o aluno.

Considerando esse cenário, Jenkins (2001) observa que o grande número de alunos que afirmam não ter competência para escrever um programa simples é alarmante.

5.1 - Classificação das Abordagens

5.1.1 - Aula Tradicional e Laboratório

Segundo Costelloe (2004b), neste tipo de ambiente o aluno na maior parte do tempo, atua de forma passiva, basicamente recebendo informações. Isto ocorre principalmente em turmas em que o número de alunos é elevado. As aulas são expositivas, seguidas de resolução de problemas em laboratório, utilizando uma linguagem de programação. Por conta disso, a ênfase muitas vezes gira em torno da linguagem de programação e não da elaboração da solução do problema, o que pode levar à dificuldades e abstrações prematuras.

A autora afirma que, nas aulas expositivas, quase não há interação entre os alunos e entre alunos e professor, além das características individuais dos alunos não desempenharem papel relevante.

Considerando nossa experiência docente, concluímos que esta metodologia não é

[5] Debug ou Depuração é o processo de encontrar erros (*bugs*) ou problemas, num aplicativo de software ou mesmo em hardware (METZGER, 2003).

adequada para o ensino de introdução à programação (algoritmos). De acordo com Costelloe (2004b), o ensino tradicional impõe uma aprendizagem baseada em regras e restringe a capacidade natural dos alunos de resolver problemas.

Segundo Linder *et al.* (2001), a educação superior tem uma tendência a priorizar os métodos de ensino tradicionais. Para comprovar, ele apresentou a seguinte tabela relacionando a modalidade de interação e a porcentagem de aproveitamento dos conteúdos pelos alunos, apresentado no Quadro 5.

Modalidade de interação	**Porcentagem de retenção do conteúdo**
1. Lendo	10%
2. Ouvindo	20%
3. Assistindo	30%
4. Assistindo e ouvindo	50%
5. Discutindo	70%
6. Fazendo e discutindo	90%
7. Ensinando e monitorando	95%

Quadro 5- Porcentagem de aproveitamento pelos alunos em relação à modalidade de interação (LINDER *et al.*, 2001)

Percebemos que os itens 6 e 7 sinalizam para que sejam modificadas as técnicas utilizadas pelo ensino tradicional. Embora esta metodologia de ensino ainda seja predominante em muitas universidades, no ensino de programação muitos professores/ pesquisadores têm procurado utilizar novas técnicas e ferramentas para facilitar a aprendizagem, tanto em aulas expositivas, quanto de laboratórios. Entre estes pesquisadores, destaca-se o grupo da Universidade de Queensland, que, em 1998, propôs um curso baseado na aprendizagem por meio da resolução de problemas e aprendizagem flexível.

A combinação de aulas expositivas com laboratórios altera este modelo para um modelo construtivista, em que os alunos podem utilizar os exercícios de laboratório para construir o conhecimento adquirido na aula expositiva. As aulas expositivas podem ser rígidas ou adotar uma postura colaborativa entre professor e aluno, na busca de soluções para os problemas propostos.

5.1.2 - Visualização de Programas

As pesquisas mais recentes em desenvolvimento cognitivo e neuropsicologia sugerem que as habilidades cognitivas são mais diferenciadas e mais específicas do que se acreditava (GARDNER, 1985). Acredita-se, hoje, que o sistema nervoso seja altamente diferenciado e que diferentes centros neurais processem diferentes tipos de informação (GARDNER, 1987).

Howard Gardner, psicólogo da Universidade de Harvard, baseou-se nestas pesquisas para questionar a tradicional visão da inteligência, uma visão que enfatiza as habilidades linguística e lógico-matemática. Segundo Gardner (1987), todos os indivíduos normais são capazes de uma atuação em pelo menos sete diferentes e, até certo ponto, independentes áreas intelectuais. Ele sugere que não existem habilidades gerais, duvida da possibilidade de se medir a inteligência por meio de testes de papel e lápis e dá grande importância a diferentes atuações valorizadas em culturas diversas. Finalmente, ele define inteligência como a habilidade para resolver problemas ou criar produtos que sejam significativos em um ou mais ambientes culturais (GAMA, 1998).

A teoria das Inteligências Múltiplas do Gardner (1993) diz que o ser humano nasce com certa porção de inteligência com áreas dominantes e outras recessivas; o mais importante é a capacidade de desenvolver qualquer área da inteligência. Os tipos de inteligência são descritos brevemente a seguir (GAMA, 1998).

1. Inteligência linguística: Os componentes centrais da inteligência linguística são a sensibilidade para os sons, ritmos e significados das palavras, além de uma especial percepção das diferentes funções da linguagem. É a habilidade para usar a linguagem para convencer, agradar, estimular ou transmitir idéias.

2. Inteligência musical: Esta inteligência se manifesta por meio da habilidade para apreciar, compor ou reproduzir uma peça musical. Inclui discriminação de sons, habilidade para perceber temas musicais, sensibilidade para ritmos, texturas e timbre, e habilidade para produzir e/ou reproduzir música.

3. Inteligência lógico-matemática: O componente central desta inteligência é a sensibilidade para padrões, ordem e sistematização. É a habilidade para explorar relações, categorias e padrões, por meio da manipulação de objetos ou símbolos, e para experimentar de forma controlada; é, ainda, a habilidade para lidar com séries de raciocínios, para reconhecer problemas e resolvê-los. É a inteligência característica de matemáticos e cientistas. Gardner,

porém, explica que, embora o talento científico e o talento matemático possam estar presentes num mesmo indivíduo, os motivos que movem as ações dos cientistas e dos matemáticos não são os mesmos. Enquanto os matemáticos desejam criar um mundo abstrato consistente, os cientistas pretendem explicar a natureza.

4. Inteligência espacial: Gardner descreve a inteligência espacial como a capacidade para perceber o mundo visual e espacial de forma precisa. É a habilidade para manipular formas ou objetos mentalmente e, a partir das percepções iniciais, criar tensão, equilíbrio e composição, numa representação visual ou espacial.

5. Inteligência cinestésica: Esta inteligência se refere à habilidade para resolver problemas ou criar produtos por meio do uso de parte ou de todo o corpo. É a habilidade para usar a coordenação grossa ou fina em esportes, artes cênicas ou plásticas no controle dos movimentos do corpo e na manipulação de objetos com destreza.

6. Inteligência interpessoal: Esta inteligência pode ser descrita como uma habilidade para entender e responder adequadamente a humores, temperamentos, motivações e desejos de outras pessoas.

7. Inteligência intrapessoal: Esta inteligência é o correlativo interno da inteligência interpessoal, isto é, a habilidade para ter acesso aos próprios sentimentos, sonhos e idéias, para discriminá-los e lançar mão deles na solução de problemas pessoais. É o reconhecimento de habilidades, necessidades, desejos e inteligências próprias; a capacidade para formular uma imagem precisa de si próprio e a habilidade para usar essa imagem para funcionar de forma efetiva. Como esta inteligência é a mais pessoal de todas, ela só é observável por meio dos sistemas simbólicos das outras inteligências, ou seja, por meio de manifestações linguísticas, musicais ou cinestésicas.

Para a aprendizagem de introdução à programação, segundo Costelloe (2004b) às habilidades espaciais têm muita importância. Entende-se por inteligência espacial a habilidade de formar um modelo mental de um mundo espacial. A visualização de programas é o mapeamento de idéias abstratas, representadas no código em representações visuais, facilitando a observação do sistema. É utilizada para auxiliar o programador ou o usuário a compreender as variáveis observadas.

De acordo com (YEHEZHEL, 2002, *apud* COSTELLOE, 2004b), 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, são a Visualização de Programas, a Animação do Algoritmo e a Programação Visual, descritas brevemente a seguir.

A Visualização de Programas é focada na representação gráfica da execução de um programa e seus dados. É possível visualizar os dados, código e os eventos desejados. Existe um baixo nível de abstração e são feitas representações diretas do código e dados. Já o usuário pode exercer um papel passivo ou ativo, fazendo as alterações por dados desejadas ao longo da execução. Entre os Visualizadores mais utilizados estão:

- **JAWAA:** Java e Animação de algoritmo baseado na *web*.
- **Alice (www.alice.org):** Tem como objetivo facilitar para os iniciantes a construção de animações gráficas em 3D. Possibilita ao aluno aprender estratégia de resolução de problemas e dá conceitos necessários para criar um programa de computador. Alice é construída utilizando a linguagem Python (www.python.org)
- **Karel:** Desenvolvido por um grupo de Waterloo. É utilizado para ensinar conceitos de orientação a objetos, seleção, interação, métodos com parâmetro e instâncias de variáveis.
- **J Karel Robot:** É uma ferramenta educacional que estende os conceitos do Robô Karel, utilizando a taxonomia de Bloom.Visa ao desenvolvimento cognitivo e ao ensino de mais conceitos presentes no currículo de cursos de informática. Seus autores acreditam que a taxonomia de Bloom provê uma base para uma maior eficácia pedagógica, em particular, identificando tópicos, exercícios e tarefas.
- **Logo:** Logo é o nome para uma filosofia educacional e uma família continuamente evoluindo de linguagens de programação para melhorar sua realização. (Abelson, 1993). Desenvolvido inicialmente há mais de 30 anos, é baseado na filosofia educacional construtivista, e foi desenvolvido como uma ferramenta de aprendizagem.

  Segundo Costelloe (2004b), baseada nos resultados das pesquisas com Logo até o momento, o potencial do Logo para desenvolver idéias geométricas deve ser utilizado para estender as atividades dos professores. Observa-se que os alunos não transferem automaticamente o conhecimento obtido com uma situação para a outra, apenas a repetição não é suficiente. Deve-se disponibilizar questões que levem os alunos a refletir sobre aquilo que estão fazendo.

Diversos projetos de ensino têm sido introduzidos na ciência da computação por meio de animações e mundo virtual. Na Universidade de Duke está sendo desenvolvido um "mundo virtual de animação", com finalidade de ensinar conceitos de ciência da computação e de programação por meio de animações simples em 2D e 3D.

Nesta universidade, os alunos se sentem motivados, trabalham em laboratório em pares. Um dado relevante a observar, é o número de alunos, neste caso quinze. As aulas expositivas (tradicionais) não foram abolidas, porém, tiveram redução e têm sido utilizadas para introduzir cada assunto a ser tratado no laboratório.

Este curso tem início com HTML, para que cada aluno possa construir seu portfólio, então o conceito de algoritmo é introduzido com exemplos simples, com a montagem de um sanduíche. A próxima etapa é a criação de animações usando JAWAA, primeiro criando vários elementos (objetos) simples e, depois, agrupando-os. Feito isso, passa-se ao aprendizado com Star Logo, então, o ambiente Alice é utilizado para criar um mundo tridimensional; finalmente, nas últimas cinco semanas, utiliza-se Karel ++ (*programming robots*).

Esta experiência levou os pesquisadores envolvidos às seguintes conclusões:

- A maioria dos alunos gosta de trabalhar em pares.
- A criação de páginas Web e o uso do programa Alice foram os tópicos preferidos por 100% dos alunos.
- Star Logo e Karel ++ ficaram em segundo lugar, com 75% da preferência dos alunos.
- JAWAA ficou com 50%. O grupo de professores está propondo utilizá-la com um editor gráfico em vez de digitar os comandos em arquivo, para o próximo curso; e,
- Alguns alunos acharam o Karel ++ muito complexo.

As avaliações mostraram que os alunos gostaram dessa modalidade de curso, mas não indicam se o curso facilitou a aprendizagem em relação ao curso de programação tradicional. O que se pode afirmar é que houve um aumento considerável na motivação dos alunos.

Quanto à **Animação de Algoritmos,** segundo Costelloe (2004b), muito esforço e recursos têm sido empregados no desenvolvimento de sistemas de animação para ensinar algoritmos em cursos de Ciência da Computação. Esses sistemas mostram diversas operações realizadas pelo algoritmo, uma vez que o usuário geralmente pode escolher o que quer visualizar e qual o nível de abstração a ser utilizado. Assim, é possível haver um alto grau de

interação entre a ferramenta e o usuário.

O primeiro sistema para animação de algoritmos foi desenvolvido por Mark Brown no início dos anos 80, chamava-se BALSA (*Brown University Algorithm Simulator and Animator*). Este sistema foi o protótipo para todos os sistemas de animação disponíveis atualmente.

Com o objetivo de avaliar empiricamente o valor pedagógico dos estudantes visualizarem e interagirem com animações de algoritmos pré-elaborados, John Stasko realizou alguns experimentos, concluindo que os benefícios dessas animações não eram óbvios (STASKO, 1996, *apud* COSTELLOE, 2004b). Os desenvolvedores concluíram que um sistema de animação de algoritmos deveria ser fácil de aprender e utilizar, além de que, o desenvolvimento das animações deveria ser intimamente ligado ao algoritmo e suas operações.

O programa Samba foi desenvolvido para atender tais requisitos e torna possível visualizar algoritmos de ordenação (*quicksort*, *heapsort*, bolha, etc.), utilizados principalmente no aprendizado de Estruturas de Dados. Da mesma forma que o Samba, vários outros programas foram desenvolvidos com a mesma finalidade.

Hundhausen (2002) realizou um estudo etnográfico integrando o sócioconstrutivismo e a visualização de algoritmos em um curso de algoritmos para alunos do nível médio. Nesse curso, os alunos eram conduzidos a construir seu aprendizado com o auxílio de ferramentas para, posteriormente, apresentar suas visualizações ao professor e a seus colegas para discussão. As técnicas de pesquisa utilizadas foram: observação participante, entrevistas informais e semi-estruturais, questionários, notas de campo, gravação de áudio e vídeo das apresentações, coleta e análise de dados. Ainda foram coletados e analisados os diários dos alunos, em que eles relatavam o processo de elaboração das animações, bem como os problemas encontrados. Os dois estudos são baseados em duas turmas ministradas pelo mesmo professor em dois trimestres consecutivos.

O estudo realizado por Haundhausen (2000) abrangeu a observação dos alunos e de um professor assistente. Ele argumenta que, de acordo com o sócioconstrutivismo, a participação na comunidade é muito produtiva. Dessa forma, a construção e a apresentação das visualizações de algoritmo, como feitas nesse curso, deram ao aluno a oportunidade de se engajar em atividades típicas do professor.

Segundo Haundhausen (2000), os exercícios motivaram os alunos a trabalharem com mais afinco e com mais motivação. Sua descoberta mais importante é que as ferramentas convencionais existentes para visualização de programas podem desviar a atenção dos alunos dos conceitos realmente importantes do curso. Ele sugere que a partir desses resultados sejam realizados rigorosos estudos empíricos para testá-los.

Com relação aos aspectos pedagógicos da utilização da Visualização de Programas, considerando que o ser humano tem diferentes estilos de aprendizagem, a visualização pode clarear conceitos complexos, possibilitando ao aluno desenvolver modelos mentais para representá-los em forma de algoritmo.

Segundo Ben Ari (2001), o construtivismo diz que cada indivíduo cria estruturas cognitivas (modelos) durante a aprendizagem. Dados sensitivos são combinados com o conhecimento existente para criar novas estruturas cognitivas, que se tornam base para outras construções. O conhecimento também é criado, cognitivamente, por meio da reflexão sob conhecimentos existentes. Ele compara a visualização com o construtivismo, pois ela auxilia na construção do conhecimento, considerando que o aluno pode controlar sua atividade e tem retorno imediato. Em ambos os casos, o conhecimento é construído de forma ativa.

Já a Programação Visual é caracterizada por Costelloe (2004b) como o uso de componentes visuais para construir um programa, sem representação textual, atrás de representações gráficas, e como exemplo podemos citar a linguagem de programação Visual Basic (VB). Humel (2002) afirmou que, apesar do *Visual Basic* ser a linguagem de programação mais popular dos EUA, não é muito utilizada nos cursos de Ciência da Computação. Uma das razões para que isso venha ocorrendo é o fato do mercado estar solicitando cada vez mais programadores em linguagem Java.

Entretanto, em algumas experiências, como na *Saint Xavier University*, pesquisadores concluíram que, para introduzir a programação, a linguagem Visual Basic seria uma opção mais adequada, se comparada com as linguagens tradicionais.

Harris (2000) relacionou as linguagens mais utilizadas em cursos de introdução à programação, cuja relação está apresentada na Quadro 6. Ele também analisou a dicotomia entre a rejeição do Visual Basic pela academia e a aceitação pela comunidade de programadores.

Linguagem	Nº de colégios / universidades	Porcentagem
C ++	12	60%
Java	5	25%
Pascal	2	10%
C	1	5%

Quadro 6 – Linguagens utilizadas em cursos de introdução à programação (HARRIS, 2000).

Outro exemplo da utilização de programação visual foi uma nova metodologia desenvolvida por Smith (SMITH, 2000, *apud* COSTELLOE, 2004b) para o ensino de programação, combinando programação por demonstração (PBD) e regras visuais antes-depois. Essa metodologia foi implementada em um sistema chamado de *Stagecast Creator*, e o sistema tornou-se um produto comercial. Inicialmente, era voltado à educação infantil, focado na simulação. Nesse sistema, o usuário demonstra os algoritmos utilizando a interface, por meio de simulações visuais, sem a necessidade de programar.

Smith e Cypher fizeram centenas de testes em escolas da Califórnia em crianças e adolescentes entre seis e doze anos, em que os professores integraram protótipos do *Stagecast Creator* ao longo do ano, com o objetivo de aprimorar a habilidade dos alunos em resolver problemas.

Pesquisadores independentes, em diversas universidades da Inglaterra, conduziram um estudo formal utilizando protótipos do *Stagecast Creator* e concluíram que os estudantes, que aprenderam programação inicialmente com o *Stagecast Creator*, tornaram-se programadores melhores do que os que aprenderam direto uma linguagem de programação.

5.1.3 - Robôs

Pesquisas na área de educação, desde Maria Montessori, que acreditava que as crianças aprendiam por meio de investigação e exploração, até Piaget, que argumentava que o conhecimento é construído por meio de operações concretas, e considerando ainda as idéias de Papert, o qual chamava para uma “re-avaliação do concreto” e afirmava que as abstrações não deveriam ser consideradas superiores às atividades concretas, todos promovem

aprendizagem via atividades físicas, colocando atividades concretas antes das abstratas. Esta linha de pensamento foi adotada por diversos pesquisadores na área de ensino de programação, que propuseram a utilização de robôs para ensinar os conceitos básicos desta disciplina.

Em 1960, o desenvolvimento do Robô Karel, citado na seção 5.1.2, deu início aos trabalhos nesta área, que avançaram muito desde então. Linder *et al.* (2001) afirmam que a aprendizagem promovida por meio da interação com um ambiente, ao invés de apenas participar de aulas expositivas, é mais efetiva. Também 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, 1996).

Posteriormente, o kit de construção LEGO foi incorporado à linguagem LOGO. Com esse kit, as crianças tinham a possibilidade de construir máquinas, ligando motores e sensores, e podiam controlá-las pelo computador. Entretanto, apenas recentemente, com a queda do custo desta tecnologia e com o desenvolvimento de Kits portáteis com LEGO/LOGO, é que foi possível a sua utilização em larga escala nas escolas e universidades.

O kit Mindstorms, desenvolvido pelo MIT (*Massachusetts Institute of Technology*), tornou-se uma ferramenta popular para o ensino, na Academia da Força Aérea dos Estados Unidos, sendo utilizado em outras universidades.

Em 2001, na Academia de Força Aérea dos Estados Unidos, foi desenvolvida uma interface para o Lego Mindstorm RCs, baseada num subconjunto da linguagem de programação Ada, chamado de Ada Mindstorms 2.0. Esta interface foi desenvolvida porque os professores envolvidos com o ensino de introdução à programação consideraram o ambiente original não adequado para um curso introdutório. O uso desta ferramenta é parte de uma experiência para ensinar programação em cursos de Ciência da Computação.

Outras experiências, utilizando robôs para o ensino de programação, obtiveram resultados satisfatórios. Para Costelloe (2004b) é necessário mais trabalho que comprovem empiricamente os benefícios do uso de robôs em relação ao ensino tradicional.

Wolz (2001) argumenta que o *feedback* imediato, fornecido pelos ambientes de desenvolvimento integrado (IOEs), deixa muitos estudantes dependentes da tecnologia. O que acontece de fato é que os estudantes passam menos tempo planejando seu trabalho, pois partem para o método da tentativa e erro. No caso dos robôs, este planejamento é

indispensável, o que leva obrigatoriamente a “pensar primeiro” ao invés de utilizar o método da “tentativa e erro”.

Wolz utilizou um kit com dois robôs LEGO para 20 (vinte) alunos, num curso experimental de programação, os quais trabalharam em grupos de quatro. No final, os estudantes responderam a um questionário sobre a experiência e seu processo de aprendizagem. Como conclusão do seu trabalho, a autora afirma que os alunos aprenderam a pensar no problema antes de programá-lo. A experiência foi considerada um sucesso pedagogicamente, e comprovou que os robôs podem ser utilizados no estágio inicial da programação e do gerenciamento de projetos, não apenas para demonstrar os programas de forma mais concreta.

Os principais benefícios, identificados por Costelloe (2004b), da utilização dos robôs são:

- Promove atividades de aprendizagem, engaja os estudantes e favorece o entusiasmo.
- Promove a colaboração e os robôs são participantes desta colaboração.
- Provê experiência com máquinas reais.
- Promove criatividade.
- Os alunos podem gerar hipóteses e testá-las, tendo um retorno imediato.
- Promove bons projetos e planejamento.
- Promove a união dos aspectos teóricos e práticos, o que facilita o aprendizado autônomo.

Com relação aos aspectos pedagógicos da utilização de robôs, podemos fazer um paralelo com as idéias da aprendizagem construtivista, em que os estudantes adquirem conhecimento construindo modelos individuais de conhecimento. Linder *et al.* (2001) acreditam que os robôs não apenas facilitam a aprendizagem construtivista, mas também provêem experiências reais, não apenas simulações, o que pode ser bastante motivador.

Linder *et al.* (2001) também observam que, para os alunos assimilarem os conteúdos ensinados, o método de “fazer e discutir” é muito eficiente. Considerando que o trabalho com robôs envolve colaboração com outros alunos, é necessário que estes representem de forma mais elaborada suas idéias abstratas, por meio de sentenças concretas, para que possam ser discutidas com os colegas, além de ser possível gerar e testar suas hipóteses, assimilando mais

informações.

5.1.4 - Aprendizagem Baseada em Problemas (PBL)

Boud and Felette (1991) definem este tipo de aprendizagem não apenas como a inserção de atividades de resolução de problemas no currículo tradicional, mas, também, como sendo o planejamento da disciplina por meio de problemas-chave na prática profissional.

O ensino é direcionado pelo problema ao invés da aula tradicional. Os problemas devem vir dos professores ou dos alunos, podendo ser bem-estruturados para os alunos iniciantes até fracamente estruturados, refletindo os problemas reais que os profissionais encontrariam no mercado.

Os alunos trabalham em pequenos grupos, com a ajuda de um facilitador; o professor não diz como resolver o problema, apenas disponibiliza recursos que possam auxiliar a resolvê-lo. Os alunos é que devem determinar o que precisam aprender nas áreas relevantes para resolver o problema.

Segundo Costelloe (2004b), no ensino da computação este tipo de aprendizagem é favorecido por alguns fatores, entre eles, o fato de a computação ser dirigida por problemas e a aprendizagem ser contínua, devido às mudanças naturais da indústria, além dos projetos em grupo serem predominantes na indústria.

Entretanto, como a introdução à programação é ministrada no início da maioria dos cursos universitários, este método não nos parece o mais adequado, devido à falta de experiência e pouco conhecimento para discernir quais pontos são relevantes ou não.

Em 1996, o Departamento de Ciência da Computação da Universidade de Sydney (Austrália), desenvolveu um projeto que abrangeu duas etapas: paralelamente a uma turma de programação convencional, foi aplicada em outra turma do mesmo período, uma versão do curso utilizando os processos PBL. Como resultado, estes alunos não acharam diferenças significantes em relação ao ensino tradicional, porém gostaram mais do curso com PBL do que os alunos convencionais (COSTELLOE, 2004b).

Devido ao sucesso que os processos PBL obtiveram na área médica, McCracken e Waters (1999) decidiram implementar um programa utilizando PBL para o ensino de engenharia de software. Para eles, esta área apresentava uma dificuldade dual pelo fato de

necessitar de um conhecimento que cresce exponencialmente para ser colocado em prática, além do grau de complexidade envolvido, que é esperado que os alunos superem.

Para compreender porque os processos PBL não tiveram o mesmo sucesso na informática do que em outras áreas, como na saúde, McCracken e Waters (1999) realizaram um estudo etnográfico dos alunos e observaram que o problema do aprendizado colaborativo está nas dificuldades em trabalhar em grupo, que fazem com que os alunos passem a evitar o problema em vez de resolvê-lo. Estes autores acreditam que é necessário, primeiramente, desenvolver habilidades para se trabalhar em equipe, além de auxiliar o aluno a identificar quais assuntos são necessários para seu aprendizado individual, aí então prosseguir com o processo PBL.

Entre os benefícios da aprendizagem baseada em problemas os autores destacam:

- Aprendizagem efetiva, de longa duração, pois ela força o aluno a refletir sobre seus processos de aprendizagem.
- Promove o entendimento por meio do trabalho colaborativo.
- A aprendizagem se dá por problemas semelhantes a problemas reais, ao invés de conteúdos.
- Os problemas podem ser bem estruturados ou fracamente estruturados, dependendo da experiência dos alunos.
- Promove criatividade em criar soluções para problemas diversificados.
- Promove habilidade de trabalhar em equipe.
- Promove o aprendizado independente e força os alunos a terem responsabilidade por seu trabalho.
- Promove sentimentos positivos em relação ao curso.
- Os alunos aprendem outras habilidades não específicas do curso. Entre essas habilidades, destacam-se a verbal, a escrita de relatório e a demonstração das soluções.

Com relação aos aspectos pedagógicos da PBL, observa-se que ela adota uma visão construtivista da aprendizagem, na qual os alunos fazem ajustes contínuos para que haja uma construção verdadeira. Segundo Costelloe (2004b), este método favorece a autonomia e as habilidades para se trabalhar em equipe, o que não tem tido a importância devida em cursos tradicionais. Além disso, engajar o aluno na construção de seu conhecimento é uma das

características da pedagogia construtivista.

5.1.5 - Aprendizado Cognitivo

A principal característica desse método é que há uma socialização do conhecimento, os problemas são selecionados de acordo com seus níveis pedagógicos e a ênfase ensinada depende do foco da aprendizagem.

Chalk (CHALK, 2000, *apud* COSTELLOE, 2004b) fez uma experiência combinando a descrição de Papert de micromundo como "um ambiente de aprendizagem ativo" e os aplicativos interativos para desenvolvimento de programas disponíveis via internet, chamando de "*webworlds*". Ele propôs em seu estudo de caso a utilização de um ambiente colaborativo BSCW (Suporte Básico para trabalho colaborativo). Nesse estudo, os alunos trabalharam em pares, cada par com uma área especifica no ambiente compartilhado, sendo que todos poderiam acessar áreas comuns. A intenção era que os alunos colocassem seus projetos para todos, de forma a compartilharem sua aprendizagem; o ambiente também podia oferecer comentários vindos de um tutor responsável por cada grupo de alunos.

Chalk (2000) utilizou observação, entrevistas, anotações dos alunos, relatório das tarefas, gravações *on-line*, questionário e testes para conduzir seu estudo de caso e concluiu que nem todas as ferramentas eram acessíveis, os modelos propostos pelos alunos eram exploratórios e o objetivo da tarefa, que era comparar a variedade de soluções dos alunos, ficou vago se comparados com o previsto. A conclusão final foi que não houve evidência de atividade criativa, além do convencional.

Entre os benefícios da Aprendizagem Cognitiva, Costelloe (2004b) destaca os seguintes:

- Promove habilidades metacognitivas, por meio da reflexão e auto-análise.
- Possibilita que alunos mais experientes dêem dicas e suporte gradualmente aos demais, quando necessário.
- Como o *feedback* é mais rápido, pode guiar os alunos.
- Os professores podem avaliar a maneira de pensar dos alunos e aplicar exercícios adequados a cada grupo.
- Promove colaboração e aprendizagem num contexto social.
- A disponibilidade de ferramentas na Web facilita o desenvolvimento da

aprendizagem; e

- Promove o desenvolvimento de outras habilidades, tais como: a verbalização e a comunicação.

Segundo Costelloe (2004b), no método de Aprendizagem Cognitiva são utilizadas as seguintes estratégias de aprendizagem:

- Modelagem, por exemplo, a demonstração de um processo de pensamento.
- Explicação, porque as atividades têm o lugar que tem.
- Os alunos podem contar com um professor praticamente particular, e com as facilidades de monitoramento, assistência e suporte às suas atividades.
- Suporte para que os alunos consigam realizar suas tarefas com o auxílio cada vez menor do professor, promovendo sua independência.
- Reflexão, auto-avaliação e auto-análise.
- Articulação, resultado da reflexão para exprimir o raciocínio verbalmente.
- Exploração, os alunos são encorajados a formar suas hipóteses e testá-las para encontrar novas idéias.

A base da aprendizagem cognitiva é como a do construtivismo, em que o aluno constrói seu conhecimento primeiro com o impulso de alguém experiente, e gradualmente vai se tornando independente. O aspecto colaborativo e o desenvolvimento de habilidades meta cognitivas se refletem em seu trabalho; assim, o aspecto social da aprendizagem, exposto por Vygotsky, é inerente neste modelo.

5.2 - Teoria Educacional

Powers & Powers (1999) salientam que nenhum método pedagógico é uma panacéia! O que deve haver é uma união de características, que, alinhadas por um objetivo, possam promover a aprendizagem. Deve-se levar em consideração aspectos sócioculturais, como número de alunos e origem, entre outros.

A teoria construtivista afirma que todo o nosso conhecimento tem de ser construído por meio de experiências pessoais e interações sociais, num conjunto cultural particular, sendo que qualquer novo conhecimento que o aluno construa em resposta a uma nova experiência, será incorporado a sua estrutura de conhecimento.

Segundo Powers & Powers (1999) para que o ensino seja efetivo, deve ser baseado na experiência. Diversas experiências didáticas aqui relatadas têm caráter experimental, por exemplo: Tecnologias Visuais, Aprendizagem Baseada em Problemas e a utilização de Robôs. Os pesquisadores envolvidos nestes trabalhos enumeram alguns requisitos para a aplicação de trabalhos em grupo, com o intuito de promover o aprendizado cooperativo, são eles:

- Objetivos individuais de aprendizagem.
- Pontuação individual.
- Independência positiva: nenhum membro do grupo deve "carregar" os demais.
- Os grupos devem ser heterogêneos.
- O professor deve colocar regras que promovam o engajamento social.
- Avaliação a *posteriori* do grupo.

Deve-se considerar os diversos modelos de aprendizagem e auxiliar os alunos para que possam realizar tarefas tanto no modelo preferido como em outro qualquer.

O modelo de indicador de tipos de Myers-Briggs (MBTI) classifica os alunos de acordo com suas preferências, numa escala derivada da teoria de Carl Jung de tipos psicológicos. Nessa escala, os estudantes podem ser:

- Extrovertidos, tentam pensar de dentro para fora, ou introvertidos, são aqueles que pensam por meio das coisas.
- Bom-senso, práticos e se orientam por detalhes, ou intuitivos, que utilizam a imaginação e se orientam pelos conceitos.
- Racionais, céticos e suas decisões são baseadas em regras, ou emocionais, apreciam as coisas ao seu redor e suas decisões são baseadas em considerações humanísticas ou pessoais.
- Julgadores, marcam e seguem sua agenda, ou perceptivos, se adaptam às mudanças circunstanciais.

As aulas tradicionais são orientadas aos introvertidos, ao passo que se desenvolvermos atividades cooperativas, privilegiamos os extrovertidos.

O modelo de estilo de aprendizagem de Kolb (1986) classifica os alunos de acordo com as seguintes preferências:

1. Experiências concretas ou abstrações conceituais e

2. Atividades de experimentação ou observação reflexiva.

Existem quatro tipos de aprendizagem neste esquema:

- Concreta / reflexiva: Neste caso, o aluno pergunta: “Por que estamos aprendendo isso?” Para ser efetivo com este tipo de aluno, o professor deve atuar como motivador.
- Abstrata / reflexiva: o aluno pergunta: “O que estamos fazendo?”. Neste caso, o professor deve atuar como um especialista.
- Abstrata / especialista: “Como podemos aplicar isso?”. O professor deve atuar promovendo um guia prático e dando os devidos retornos.
- Concreto / ativo: “O que se...”. O professor deve atuar possibilitando a experiência com novas situações e auto-aprendizagem.

Em todos esses casos, o professor deve deixar claro a relevância de cada tema abordado, encorajando a exploração. Tem-se ainda o modelo de Felder-Silverman (1988), que classifica os alunos da seguinte forma:

- Sensorial: concreto e prático, ou intuitivo: conceitual, inovador e orientado pela teoria.
- Visual ou verbal.
- Indutivo: trabalha do específico para o geral, ou dedutivo: trabalha do geral para o específico.
- Ativo: vai além do conteúdo abordado e trabalha com os outros, ou reflexivo: é aquele que pensa nas coisas por si só, do começo ao fim.
- Sequencial: trabalha de forma linear, global: aprende em pequenos passos.

  Felder (1988) afirma que o professor deveria trabalhar essas formas de maneira alternada, utilizando a mais adequada a cada situação a ser ensinada; porém, tanto os alunos como os professores têm a tendência de utilizar sua forma preferida.

Quando adotamos novos métodos para ensinar programação, devemos ter em mente os diferentes estilos de aprendizagem e buscar um equilíbrio entre eles, por exemplo, se focarmos a visualização, então devemos complementar com o método verbal ou explicações escritas.

A teoria das inteligências múltiplas do Gardner (1993) afirma que existem diversos tipos de inteligência, cada pessoa é dotada de um nível variável de cada uma dessas

inteligências. Na área de computação, espera-se ver estudantes com alto nível de inteligência lógica; entretanto, eles podem e devem aplicar outras inteligências nas diferentes áreas do curso.

A sequência em que é ministrado o conteúdo tem um papel importante, e, muitas vezes, é determinada pelos livros adotados. Devemos ser cautelosos ao estabelecer esta sequência e, segundo Costelloe (2004b), é possível nos basearmos na taxonomia de Bloom da aprendizagem por objetivos, indo de um nível de conhecimento simples a processos mais elaborados.

Lister (LISTER, 2000) acredita que os alunos não devem começar escrevendo programas diretamente, mas, sim, que devem ser utilizados níveis de aprendizagem. É possível iniciar o currículo partindo de tópicos mais concretos para os mais abstratos, seguindo as idéias de Piaget. A sequência sugerida é iniciar a programar com objetos concretos, neste caso os robôs poderiam ser úteis, para, posteriormente, trabalhar com ambientes abstratos.

Powers & Powers (1999), seguindo os argumentos de Papert, acreditam que o significado atribuído ao termo concreto deveria ser revisto e argumentam que o raciocínio abstrato não deve ser visto como mais valioso do que as manipulações concretas.

Independente da metodologia utilizada, devemos buscar o aprimoramento do ensino, para que o mesmo possa promover a aprendizagem significativa. É importante considerar os aspectos pedagógicos, prestar a devida atenção ao público e levar em consideração a variedade de estilos de aprendizagem existentes.

Baseados nos trabalhos de Costelloe (2004a, 2004b), chegamos a algumas conclusões, expostas a seguir.

A comunidade acadêmica reconhece a necessidade de complementar a abordagem tradicional com outras tecnologias; as pesquisas são baseadas numa pedagogia de fundo, por exemplo, variações do construtivismo. Porém, a falta de evidências empíricas de como essas abordagens poderiam ser incorporadas no currículo e também quanto à eficácia das mesmas têm inibido a sua utilização. Ou seja, segundo Costelloe (2004b) é necessário que sejam realizadas um número maior de pesquisas exploratórias, para que sejam disponibilizados dados concretos a respeito do desempenho das abordagens em questão.

Alguns aspectos devem ser considerados em abordagens ou ferramentas, com relação

a sua eficácia, são eles:

- Envolver o aluno com as atividades de aprendizagem.
- Permitir ao aluno um certo grau de controle sobre seu aprendizado, possibilitando sua progressão e revisão.
- Prover mecanismos de “escalada”, por exemplo: explicações, ajuda sensível ao contexto, estudo de caso e criação de padrões.
- Fornecer *feedback* em tempo adequado.
- Utilizar a colaboração.
- Incorporar mecanismos de auto-análise para promover habilidades de aprendizagem metacognitiva e independente.
- Apresentar múltiplas representações de conceitos e textos.

A maioria das ferramentas baseadas em tecnologia é voltada para visualização de algoritmos e programas já prontos. Algumas poucas são voltadas para quem está começando a aprender programação. Ziegler e Crews (1999) apontam para a necessidade de mais pesquisas para avaliar a etapa em que o aluno entende o processo de conceber uma solução computacional, independentemente da linguagem que será utilizada posteriormente, o que vai ao encontro do nosso trabalho.

Segundo Winslow (1996), estudos psicológicos sobre programação concluíram que os iniciantes, apesar de conhecerem a sintaxe e a semântica de comandos individualmente, têm muita dificuldade para combiná-los de forma correta para solucionar um dado problema, mesmo quando eles conhecem a solução informal (intuitiva).

Com os dados apresentados nestes trabalhos, é possível optar por uma abordagem, de acordo com o objetivo da aprendizagem em questão.

Costelloe (2004b) finaliza sua discussão enfatizando o papel do professor, não mais como detentor do saber, senão como um facilitador do ensino/aprendizagem entre os alunos e os especialistas, e destacando a importância de que as tecnologias citadas não são mutuamente exclusivas, e devem ser combinadas para obter uma melhor aprendizagem.

Considerando os resultados obtidos por estes grupos de pesquisa, observamos que os alunos que foram beneficiados com as metodologias empregadas foram aqueles que conseguiam conceber o raciocínio computacional, estes avançaram mais do que com as

formas de ensino tradicional; entretanto, aqueles que tinham dificuldades em elaborar o raciocínio computacional permaneceram na mesma situação.

Com isso, reforçamos nossa hipótese de que as dificuldades dos alunos estão ligadas ao processo de discretização do raciocínio, necessário à elaboração do raciocínio computacional, em virtude da abstração envolvida neste processo, e não apenas às questões de cunho didático. Buscando validar esta hipótese, decidimos realizar um estudo empírico, na forma de um estudo de caso descrito nos próximos capítulos.

CAPÍTULO 6

Procedimentos Metodológicos

6.1 – Estratégia da Pesquisa

Com base em nossas reflexões teóricas, sentimos a necessidade de confirmar nossas hipóteses pela verificação *in loco* do processo de aprendizagem dos alunos. Utilizamos como estratégia de pesquisa a metodologia do Estudo de Caso, com múltiplos casos. Segundo Yin (2005), o Estudo de Caso deve ser a estratégia escolhida para examinar acontecimentos contemporâneos no seu contexto real. Neste tipo de pesquisa, as fronteiras entre o fenômeno e o contexto não são evidentes; além disso, múltiplas fontes de evidências podem ser utilizadas.

Nosso campo para coleta de dados foram três turmas, com aproximadamente 25 alunos, da disciplina de Lógica de Programação, ministradas sempre no primeiro semestre do Curso de Tecnologia de Sistemas para Internet, da Universidade Tecnológica Federal do Paraná (UTFPR). Os instrumentos utilizados para coleta de dados foram três provas, referentes à primeira parte da disciplina, em que o conteúdo abordado abrangia estruturas seqüenciais, de decisão, controle e repetição.

Em virtude de nosso instrumento de coleta de dados serem provas, optamos por não realizar o scaneamento das mesmas, pois o processo de correção e revisão, ao qual este tipo de instrumento é submetido, pode desconfigurar a solução proposta pelo aluno.

As análises realizadas foram feitas de forma qualitativa, considerando cada aluno individualmente, com o objetivo de verificar todas as alternativas possíveis de solução. Os alunos foram apelidados de A1, A2, A3 e, assim, sucessivamente. Depois de realizar a análise da prova de cada aluno, procuramos estabelecer categorias em que as soluções encontradas pudessem ser agrupadas em cada caso.

Para apresentar os resultados dessas análises, os dados obtidos foram sistematizados de modo a facilitar a leitura e a interpretação dos mesmos.

6.2 – Contexto da Disciplina de Lógica de Programação

A disciplina de *Lógica de Programação* foi elaborada pela comissão responsável pela

criação do Curso de Tecnologia em Informática, em 1999, da qual fizemos parte; chamado a partir de 1998, de Curso de Tecnologia em Sistemas para Internet.

Esta disciplina tem como objetivo desenvolver o raciocínio computacional, para elaborar algoritmos, por meio de uma linguagem escrita (Portugol) e uma gráfica (Fluxograma) seguindo os preceitos da programação estruturada[6]. O conteúdo ministrado é composto por uma parte introdutória de lógica matemática que será utilizada na elaboração dos algoritmos. O Apêndice II apresenta uma versão do plano de ensino (Apêndice II.1) e do plano de aula (Apêndice II.2) da disciplina de Lógica de Programação, compatível com as versões utilizados nos semestres que compõem os estudos de caso, descritos no Capítulo 7 deste trabalho.

O Curso de Tecnologia de Sistemas para Internet teve início no segundo semestre de 1999. Em sua primeira versão, esta disciplina era composta por 50% de lógica matemática e 50% de algoritmos; depois de alguns ajustes, o conteúdo de lógica matemática foi reduzido para 25%, ficando 75% para o desenvolvimento de algoritmos. As principais modalidades de iteração utilizadas nesta disciplina, considerando aquelas apresentadas por Linder *et al.* (2001), apresentadas no Quadro 5, são a 4-Assistindo e ouvindo e a 6-Fazendo e discutindo. Na modalidade 6, os alunos realizam exercícios ora com a ajuda do computador, em laboratório, ora utilizando lápis e papel, em sala de aula convencional.

A forma de avaliação era composta por três provas e por listas de exercícios realizados ao longo de semestre. Até o semestre 02/2004, o conteúdo avaliado na primeira prova era composto por exercícios de lógica matemática, ficando a parte referente ao desenvolvimento de algoritmos para as outras duas provas. Pela nossa experiência e pesquisas, fomos implementando mudanças ao longo de cada semestre, com o objetivo de melhorar a aprendizagem.

A partir do semestre 01/2005, o conteúdo básico de lógica matemática passou a ser responsável por 50% da avaliação em questão. Durante os semestres que se seguiram, o conteúdo de algoritmo foi ganhando espaço. A cada semestre era realizada uma tentativa, buscando um melhor aprendizado dos alunos, na disciplina como um todo. Inicialmente, esta primeira avaliação abordava os comandos de entrada e saída de dados, operadores aritméticos

---

[6] Programação estruturada é uma metodologia para desenvolvimento de programas. De acordo com esta metodologia, as estruturas utilizadas na construção de um programa devem se restringir às estruturas seqüencial, de decisão e de iteração, descritas na seção 4.6. Além disso, os programas devem ser construídos de forma modular e hierárquica, em que o programa deve ser dividido em programas menores (módulos).

e relacionais, estruturas seqüenciais e de seleção. Tais conceitos são bastante familiares para os alunos, pois se assemelham aos conceitos matemáticos elementares.

No semestre 01/2006, optou-se por incluir problemas envolvendo estruturas de repetição, nos quais os alunos com freqüência apresentam problemas de aprendizagem, em virtude do processo de discretização inerente a essas estruturas (SETTI, 2007). Como esperado em função dos nossos estudos, o desempenho dos alunos, com a inclusão da estrutura de repetição, não foi o mesmo obtido nos semestres anteriores, como mostra o Quadro 7. O que nos levou, a no semestre seguinte, deixar o conteúdo de repetição para uma próxima fase.

Entretanto, a partir do semestre 01/2007, incluímos definitivamente as estruturas de repetição na primeira fase da disciplina. Para melhorar o processo de aprendizagem dessas estruturas, realizamos algumas modificações na forma como elas vinham sendo abordadas. Entre essas modificações, destacam-se:

1. Inclusão de fluxogramas para elaborar os algoritmos.
2. Resolução de um mesmo problema por meio de duas versões de algoritmos, a primeira utilizando estruturas sequenciais e a segunda por meio de estruturas de repetição.
3. Aumento do número e da variedade de exercícios propostos.

Com isso, observamos no semestre 02/2007, uma melhora na aprendizagem, motivada por mudanças pedagógicas, além dos tipos de registros de representação utilizados.

Avaliações com estrutura de repetição				**Avaliações sem estrutura de repetição**		
Semestre	01/2006	01/2007	02/2007	01/2005	02/2005	02/2006
Média da 1ª Prova	7,2	6,1	7,1	8,4	7,6	9,1

Quadro 7 - Média obtida na primeira prova de cada semestre

Outro fator relevante é que até o semestre 01/2007 não era utilizada nenhuma ferramenta informatizada de apoio ao desenvolvimento de algoritmos. Os testes eram feitos pela técnica manual, chamada de “teste de mesa”. Nessa técnica, o algoritmo é “executado”, por meio de dados de entrada aleatórios, via lápis e papel, linha por linha, para verificar qual

será o resultado obtido.

No semestre 01/2007, fizemos uma avaliação, junto com os alunos, de diversas ferramentas para o ensino de algoritmos disponíveis na Internet. Entre elas, as que mais nos chamaram a atenção foram os compiladores de pseudocódigo Visualg (SOUZA, 2004) e Portugol IDE (MANSO, 2005).

Essas ferramentas permitem ao aluno editar o algoritmo, fazer a correção sintática e semântica (compilação) e, então, executá-lo. O mais importante é a possibilidade de acompanhar a sua execução passo a passo (linha por linha), para monitorar o conteúdo das variáveis. Este processo é similar ao teste de mesa, citado anteriormente, porém, mais dinâmico e estimulante para o aluno.

Assim como em algumas experiências relatadas por Costelloe (2004b), observamos que os alunos se sentiram mais motivados para resolver os exercícios propostos, uma vez que tinham a possibilidade de encontrar e corrigir imediatamente seus erros, para, então, executar o algoritmo e conferir seu funcionamento. Entretanto, deve-se ressaltar que os alunos com dificuldades em conceber o raciocínio computacional, relacionadas ao processo de discretização do pensamento, não apresentaram uma melhora significativa no nível de aprendizagem.

Não faz parte do escopo deste trabalho uma avaliação mais detalhada sobre o impacto da utilização de ferramentas na aprendizagem de algoritmo. Nosso interesse neste momento é o de proporcionar alternativas que possam auxiliar os alunos a superar as dificuldades em elaborar e desenvolver o raciocínio computacional. Sugerimos, para o futuro, a realização de uma pesquisa específica para analisar este aspecto da aprendizagem.

6.3 – Estratégias de Ação

Na condição de professora da disciplina de Lógica de Programação, desde 02/00, e considerando o número de alunos das turmas, aproximadamente 20, foi possível utilizarmos uma metodologia diferenciada, em que a interação professor/aluno, bem como aluno/aluno, foi privilegiada.

A disciplina era ministrada por meio da seguinte rotina:

- Resgate dos tópicos abordados previamente.
- Apresentação do assunto a ser abordado no encontro atual.

- Proposição de exercícios envolvendo os tópicos abordados, desde o início, até o momento, pois para construção de algoritmos, os novos recursos são acrescentados aos anteriores.
- Os alunos podiam trabalhar em dupla, ou de forma individual, sendo que a discussão entre eles era estimulada pelo professor com o objetivo de trocar estratégias de soluções.
- Durante a solução dos exercícios, o professor atendia os alunos individualmente, discutindo com cada um sua idéia de solução.
- Quando os alunos terminavam suas construções, o professor ora convidava-os para que anotassem suas soluções no quadro, ora apresentava, ele mesmo, uma proposta de solução e solicitava aos alunos que encontrassem formas alternativas de resolver a questão.
- Quando se iniciou a utilização do Portugol IDE, em sala de aula, o professor continuou com a mesma sistemática de acompanhamento. A principal alteração foi que, além das dificuldades de encontrar uma solução, os alunos precisavam estar mais atentos às regras de sintaxe, para que a ferramenta pudesse executar o algoritmo.
- As unidades de análise para realizar o estudo de caso deste trabalho foram avaliações individuais, por meio de provas escritas, tanto via lápis e papel, quanto via ambiente informatizado.

Considerando a diversidade de fatores que influenciam o processo de aprendizagem dos alunos, planejamos um estudo de caso, com múltiplos casos, em que cada caso corresponde a um aspecto da aprendizagem a ser analisado, buscando subsídios para avaliar nossas hipóteses.

Este trabalho tem as seguintes hipóteses, previamente apresentadas no Capítulo 1:

1. A discretização do raciocínio, ou seja, a elaboração de um raciocínio de forma discreta para um processo de natureza contínua, é um obstáculo epistemológico à aprendizagem.
2. A identificação e manipulação da estrutura de repetição, que estão diretamente relacionadas a essa discretização, são obstáculos epistemológicos para a aprendizagem de algoritmos.
3. A passagem de um tipo de registro de representação semiótica, a saber, a linguagem

natural, para outro, a linguagem algorítmica, comporta um obstáculo à aprendizagem de algoritmos.

O primeiro caso a ser apresentado, na seção 7.1, teve como objetivo verificar as dificuldades dos alunos, em relação à aprendizagem da estrutura de repetição e seu papel como obstáculo epistemológico, buscando subsídios para avaliar as hipóteses 1 e 2. O segundo caso, apresentado na seção 7.2, teve como objetivo verificar as dificuldades dos alunos, em relação aos registros de representação, utilizados no ensino/aprendizagem de algoritmos, buscando subsídios para avaliar a hipótese 3.

O último caso, apresentado na seção 7.3, teve como objetivo verificar a relação entre a elaboração do raciocínio matemático e do raciocínio computacional pelos alunos, buscando verificar se aqueles alunos que elaboraram o raciocínio matemático de forma coerente para resolver o problema conseguem elaborar também o raciocínio computacional correspondente, na forma de um algoritmo. Neste caso, todas as hipóteses foram consideradas.

CAPÍTULO 7

Estudos de Caso no Curso de Tecnologia em Informática – UTFPR

A aprendizagem da estrutura de repetição foi escolhida para o primeiro estudo de caso, considerando o histórico do desempenho dos alunos na disciplina de Lógica de Programação, descrito na seção 6.2, bem como, os resultados obtidos por Barbosa (2001), que indicam que os alunos apresentaram dificuldades em formalizar as estruturas de seleção, e que essas dificuldades se agravavam quando o processo de repetição estava presente.

7.1 – Análise da Avaliação 01/2007

Este caso teve como objetivo verificar as dificuldades dos alunos, em relação à aprendizagem da estrutura de repetição e seu papel como obstáculo epistemológico, buscando subsídios para confirmar, ou não, as hipóteses 1 e 2.

O instrumento a ser analisado é uma avaliação, por meio de uma prova individual, da primeira parte dessa disciplina de Lógica de Programação. A avaliação completa pode ser visualizada no Apêndice I-1.

A prova em análise era composta por duas partes, a primeira referente ao conteúdo inicial de lógica matemática e a segunda abordava o conteúdo de algoritmos (comandos de entrada e saída de dados, operadores aritméticos e relacionais, estruturas sequenciais, de seleção e de repetição).

Neste estudo, nos limitaremos a analisar a primeira questão da segunda parte, em que já foi possível identificarmos o problema procurado. A seguir, a descrição da questão:

Questão 1- 01/07) Faça um algoritmo para calcular a média dos alunos de sua turma (máximo 30) no semestre 01/2007. O algoritmo deve:

- Ler as notas das 7 disciplinas do primeiro período de cada aluno, fazendo a consistência para que as notas sejam ≥ 0.
- Calcular a média de cada aluno, e escrever na tela.
- Encontrar a maior média da turma.
- Verificar o percentual de alunos que tiveram alguma reprovação e, ao final, escrever o valor na tela.

Ao estipular o número de alunos igual a trinta, a idéia é forçar o aluno a pensar em uma estrutura de repetição. Da mesma forma, foram estabelecidas sete disciplinas para que houvesse outra estrutura de repetição. Além disso, para fazer a consistência para que a nota fosse ≥ 0, haveria necessidade de uma terceira estrutura de repetição.

Considerando estes aspectos, apresentamos, a seguir, uma alternativa de solução para o problema proposto, apenas para ilustrar a nossa discussão. Esta solução faz uso das estruturas de repetição, conforme descrito no parágrafo anterior. Observamos que poderiam ser utilizadas outras estruturas de repetição, como as descritas no item 4.6.2.

Solução para a questão 1- 01/07:

```
inicio          // início do algoritmo em portugol
inteiro Num-Alunos, Notas;
real N, Média, Maior-Média, Rep;
Num-Alunos ← 0; Notas ← 0;
N, Média ← 0; Maior-Média ← 0; Rep ← 0;
N ← -1;
enquanto (Num-Alunos ≤30)
inicio
        enquanto (Notas ≤ 7)
        inicio
                escreva (“Entre com a nota do aluno, deve ser ≥ 0”);
                enquanto (N < 0)
                inicio
                        leia N;
                        escreva (“A nota deve ser ≥ 0, digite novamente”);
                fim-enquanto
                Média ← Média + N;
                Notas ← Notas + 1;
        fim-enquanto
        Média ← (Média / 7);
        escreva (“A média do aluno é = ”,  Média);
        se (Média > Maior-Média)
                entao Maior-Média ← Média;
```

se (Média < 7,0)
entao Rep ← Rep + 1;
Num-Alunos ← Num-Alunos +1;
fim-enquanto
escreva (“A maior média é = ”, Maior-Média);
escreva (“O percentual de alunos com reprovações é = ”, (Rep * 100/ 30));
fim.

Os alunos foram nominados de A1, A2, A3 ... A23. Com base nas soluções encontradas, estabelecemos as seguintes categorias:

A: O aluno utilizou a estrutura de repetição da forma esperada: utilizou a estrutura de repetição adequadamente, ou seja, de forma similar à solução anteriormente apresentada.

A’: Realizou o cálculo corretamente: realizou o cálculo corretamente.

B: Sequencial: não utilizou a estrutura de repetição, utilizou apenas estruturas sequenciais.

C: Fez para um aluno: fez o processo corretamente, porém, para um aluno. Não utilizou as estruturas de repetição.

D: Não realizou: não realizou o que foi solicitado.

E: Incorreto: elaborou o algoritmo de forma incorreta.

F: Incompleto: elaborou o algoritmo de forma incompleta.

Para melhor leitura desse trabalho, sintetizamos os dados, ilustrados no Quadro 8, em que os itens solicitados na questão correspondem às colunas e as linhas representam as categorias de soluções encontradas.

Itens/ Categorias	**R1- (30 alunos)**	**R2- (7 discipl.)**	**R3- (Notas > 0)**	**Cálculo da média**	**Maior Média**	**% de reprov.**
A: O aluno utilizou a estrutura de repetição da forma esperada	11	6	4			
A’: Realizou o cálculo corretamente				15	9	10
B: Sequencial	0	11	9			
C: Fez para um aluno	7	0	0			
D: Não realizou	0	0	9	3	13	11

E: Incorreto	3	4	1	3		
F: Incompleto	2	2	0	2	1	2

Quadro 8- Síntese geral das soluções propostas, pelos alunos, para a questão1 – 01/07

Além das categorias relativas às soluções propostas pelos alunos, foi importante considerar o conhecimento prévio em programação desses alunos. Com essa finalidade, acrescentamos uma coluna referente à experiência, que mostra se o aluno já cursou a disciplina, se sim, quantas vezes ou se trabalha na área de programação. Adotamos a seguinte notação:

R(*k*): alunos que cursaram anteriormente a disciplina *Lógica de Programação*, em que *k* representa o número de vezes.

Exp: alunos que possuíam experiência em programação.

As soluções individuais dos alunos estão ilustradas no Quadro 9.

Itens/ Alunos	Experi-ência	R1- (30 alunos)	R2- (7 discipl.)	R3- (Notas > 0)	Cálculo da média	Maior Média	% de reprov.
A1		A	B	B	A’	A’	A’
A2		A	A	A	A’	A’	A’
A3		A	A	B	A’	A’	A’
A 4		A	B	B	A’	A’	A’
A5		A	A	A	A’	A’	A’
A6		A	B	D	A’	A’	A’
A7		A	A	A	A’	A’	A’
A8		A	A	D	A’	A’	A’
A9 *		A	B	D	F	F	F
A10		A	B	B	A’	A’	A’
A11	R(3)	F	F	D	D	D	D
A12		C	B	B	A’	D	A’
A13		F	F	D	F	D	D
A14		C	B	B	A’	D	D

Itens/ Alunos	Experi-ência	R1- (30 alunos)	R2- (7 discipl.)	R3- (Notas > 0)	Cálculo da média	Maior Média	% de reprov.
A15		C	B	A	A’	D	F
A16	R(4)	E	E	D	D	D	D
A17		E	E	D	E	D	D
A18		C	B	B	A’	D	D
A19		C	B	B	A’	D	D
A20		C	B	B	A’	D	D
A21**		C	E	E	E	D	D
A22		E	E	D	E	D	D
A23	R(3)	A	A	D	D	D	D

Quadro 9 – Análise individual da solução proposta pelos alunos para a questão 1 – 01/07

A9* - Como o aluno inicializou as variáveis dentro dos laços, não chegaria aos resultados desejados, embora tenha elaborado os cálculos corretamente.

A21** - Utilizou a linguagem para representar algoritmos (*Portugol*) incorretamente, não possuía o conceito correto de variável e das operações elementares.

Analisando o Quadro 9, verificamos que dos vinte e três alunos, em relação à estrutura de repetição R1, apenas onze resolveram por meio de estruturas de repetição; em relação à estrutura R2 este número diminuiu para seis, e na R3, para quatro.

Os resultados obtidos com este estudo caso reforçaram nossas hipóteses de que a estrutura de repetição é um obstáculo epistemológico à aprendizagem de algoritmos. Os alunos não conseguem realizar o processo de discretização, necessário para transformar o raciocínio intuitivo, de natureza contínua, no correspondente computacional, de natureza discreta, como pode ser visto no exemplo de solução a seguir.

Exemplo de obstáculo à realização do processo de discretização:

```
inicio          // início do algoritmo em portugol
inteiro Nota1, Nota2, Nota3, Nota4, Nota5, Nota6, Nota7 ;
real Média, Maior-Média, Rep;
Média ← 0; Maior-Média ← 0; Rep ← 0;
escreva (“Entre com as notas do aluno, devem ser ≥ 0”);
```

leia (Nota1, Nota2, Nota3, Nota4, Nota5, Nota6, Nota7);
se ((Nota1 < 0) **ou** (Nota2< 0) **ou** (Nota3< 0) **ou** (Nota4< 0) **ou** (Nota5< 0) **ou** (Nota6< 0) **ou** (Nota6< 0))
entao Escreva (“A nota deve ser ≥ 0”);
Média ← ((Nota1+Nota2+Nota3+Nota4+Nota5+Nota6+Nota7) /7);
escreva (“A média do aluno é = ”, Média);
se ((Nota1 < 7) **ou** (Nota2< 7) **ou** (Nota3< 7) **ou** (Nota4< 7) **ou** (Nota5< 7) **ou** (Nota6< 7) **ou** (Nota6< 7))
entao escreva (“O aluno possui reprovação”);
fim.

Considerando o Quadro 9, dos vinte e três alunos analisados, onze propuseram a solução neste formato. Neste tipo de solução, o algoritmo realiza o cálculo da média para um aluno, ao invés de trinta, como pedia o enunciado. Isto se deve ao número trinta ser, propositalmente, um número grande, para utilizar o processo análogo ao realizado para a leitura das notas, caso contrário, provavelmente esses alunos realizariam a leitura de forma sequencial.

Com relação à consistência para garantir que as notas fossem maior que zero, observa-se que a utilização da estrutura condicional, apenas informa que uma das notas é inválida, mas continua a execução do algoritmo, embora os dados estejam incorretos. Com a utilização da estrutura de repetição, o algoritmo garante que as notas sejam maior que zero, para então prosseguir.

Quanto ao item correspondente ao cálculo da média de cada aluno, o processo de leitura das notas, realizado por meio de estruturas sequenciais, leva ao resultado correto, porém de forma mais trabalhosa.

7.2 – Registros de Representação Utilizados para Elaboração de um Algoritmo

O objetivo deste caso, foi o de analisar as dificuldades dos alunos, com relação aos registros de representação, utilizados no ensino/aprendizagem de algoritmos, considerando alguns aspectos abordados na seção 3.3 deste trabalho, buscando subsídios para confirmar, ou não, nossa terceira hipótese.

Este caso não englobou a estrutura de repetição, propositalmente, pois o foco foi o desempenho dos alunos face aos registros de representação. Para tal, a estrutura de seleção

simples e composta se mostrou bastante adequada.

O instrumento a ser analisado é uma avaliação, por meio de uma prova individual, com duas questões da disciplina de Lógica Aplicada. A avaliação completa pode ser visualizada no Apêndice I-2. A prova em análise era composta por duas partes, a primeira referente ao conteúdo inicial de lógica matemática e a segunda, com duas questões, abordava os seguintes conteúdos sobre algoritmos: comandos de entrada e saída de dados, operadores aritméticos e relacionais, estruturas sequenciais e de seleção.

A seguir, a descrição da primeira questão avaliada:

Questão 1 – 02/07) Faça um algoritmo que leia o ano de nascimento de uma pessoa, calcule e mostre sua idade, considerando o ano atual. Verifique e mostre se ela já tem idade para votar (idade > 16), e para tirar Carteira de Habilitação (idade > 18). O algoritmo deve emitir as mensagens adequadas, informando as possibilidades do usuário. Faça o algoritmo primeiro identificando as entradas, saídas e o processamento (E/S/P) e, depois, em portugol.

A idéia da verificação da idade para votar e dirigir, era para sugerir um encadeamento das condições, embora o mesmo não seja obrigatório. A solução esperada está posta a seguir.
Solução para a questão 1 - 02-2007:
Identificação das entradas, saídas e processamento (E/S/P):

E: Ano de Nascimento.

S: Mensagens com possibilidades do usuário.

P: Calcular a idade, comparar com a idade mínima para dirigir e votar.

Considerando que existem diversas formas de solucionar o problema proposto, mostraremos duas versões, que correspondem à maioria das respostas observadas. Na primeira versão, o algoritmo foi elaborado utilizando encadeamento das condições.

```
inicio          // início do algoritmo em portugol – primeira versão
inteiro Ano Nascimento, Ano Atual, Idade;
Idade ← 0; Ano Atual ← 2008;
escreva (“Entre com o ano do seu nascimento:”);
leia Ano Nascimento;
Idade ← (Ano Atual – Ano Nascimento);
se (Idade > 18)
```

```
    entao escreva("Você tem idade para votar e dirigir.")
    senao se (Idade > 16)
                entao escreva ("Você tem idade para votar! Mas não tem idade para dirigir.")
                senao escreva ("Você não tem idade para votar, nem para dirigir.")
fim.
```

Na segunda versão, o algoritmo foi elaborado utilizando as condições de forma sequencial.

```
inicio          // início do algoritmo em portugol – segunda versão
inteiro Ano Nascimento, Ano Atual, Idade;
Idade ← 0; Ano Atual ← 2008;
escreva ("Entre com o ano do seu nascimento:");
leia Ano Nascimento;
Idade ← (Ano Atual – Ano Nascimento);
se (Idade < 16)
        entao escreva ("Você não tem idade para votar, nem para dirigir.")
se (Idade > 16) e  (Idade < 18)
        entao escreva ("Você tem idade para votar! Mas não tem idade para dirigir.")
se (Idade > 18)
        entao escreva("Você tem idade para votar e dirigir.")
fim.
```

O algoritmo representado por meio de um fluxograma com as condições encadeadas, está ilustrado na Figura 2. A diferença entre fazer o encadeamento ou não está no fato de que, ao realizar o encadeamento, o algoritmo executa o comando vinculado à condição satisfeita e sai da condicional. Ao passo que, se as condicionais estão postas de forma sequencial, mesmo sabendo que apenas uma das condições vai ser satisfeita, todas elas serão testadas desnecessariamente.

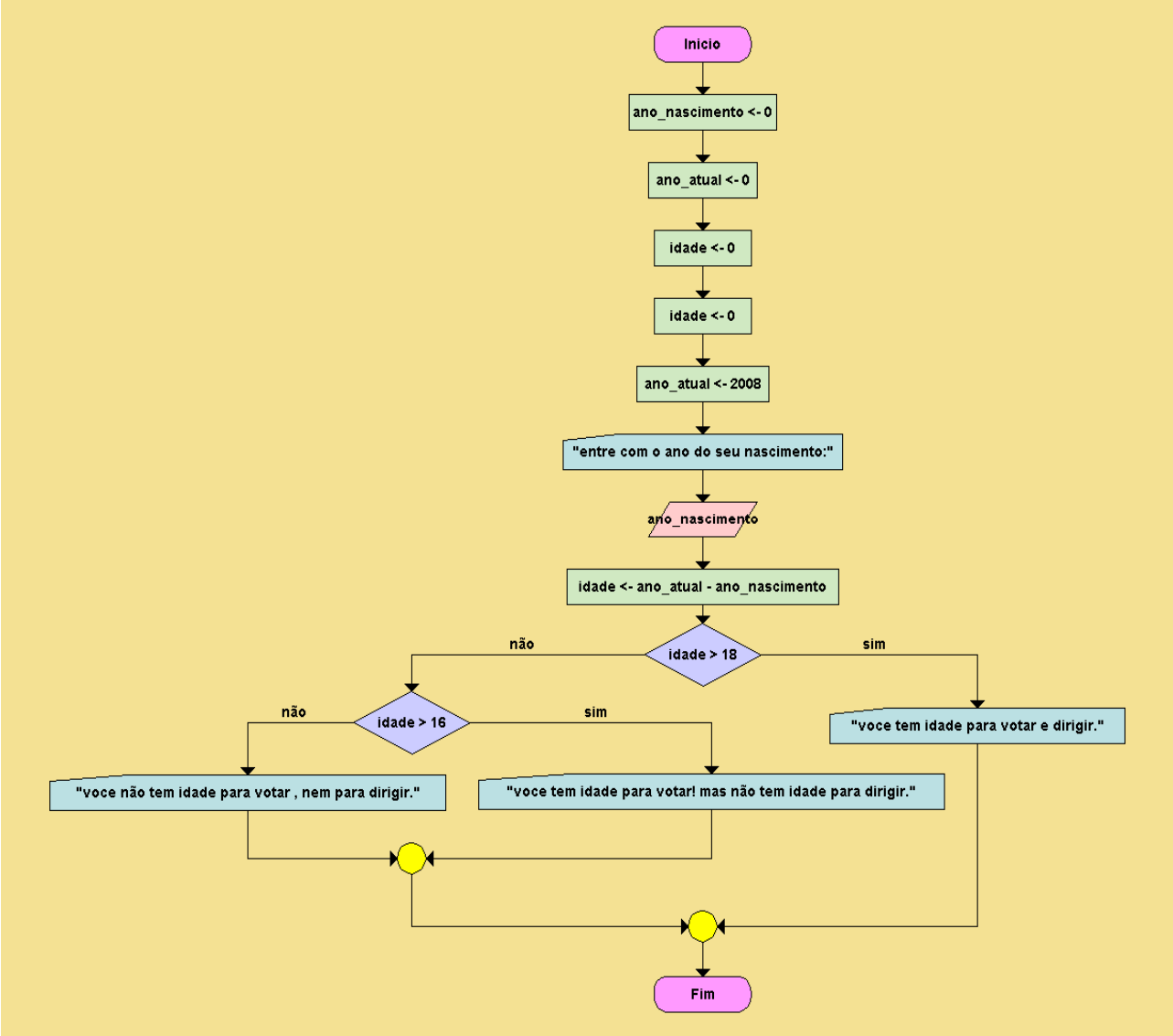


Figura 2 – Algoritmo representado em fluxograma para a questão 1 – 02/2007

Fonte: A autora.

Os alunos foram nominados de A1, A2, A3 ... A27, com base nas soluções encontradas, e estabelecidas as seguintes categorias:

A: Correto: o aluno identificou as entradas, saídas e o processamento corretamente.

B: Fluxograma: resolveu o problema utilizando fluxograma, ao invés de portugol.

C: Não realizou: não realizou o que foi solicitado.

D: Sequencial: fez o processo de forma seqüencial.

E: Encadeado: implementou as estruturas de seleção de forma encadeada, de acordo com a solução anteriormente apresentada.

F: Incorreto: elaborou o algoritmo incorretamente.

Para melhor leitura desse trabalho, sintetizamos os resultados, que estão ilustrados no Quadro 10, em que os itens solicitados na questão correspondem às colunas, e as linhas representam as categorias de soluções encontradas.

Itens/ Categorias	Identificar E/ S/ P	Portugol
A: Correto	25	
B: Fluxograma	1	
C: Não realizou	1	1
D: Sequencial		13
E: Encadeado		12
F: Incorreto		1

Quadro 10 – Síntese geral da solução proposta pelos alunos para a questão 1 – 02/07

Da mesma forma que no estudo de caso anterior, apresentado na seção 7.1, o conhecimento prévio em programação dos alunos foi considerado. Para tal, acrescentamos a coluna referente à experiência, que mostra se o aluno já cursou a disciplina; se sim, quantas vezes ou se trabalhava na área de programação. Utilizamos a mesma notação da seção 7.1.

As soluções individuais dos alunos estão ilustradas no Quadro 11.

Itens/ Alunos	Experiência	Identificar E/ S/ P	Portugol
A1	R(3)	A	D
A2		A	D
A3		A	E
A4	R(2)	A	D
A5		A	D
A6		A	E
A7		A	D
A8		A	D
A9		A	D
A10		A	E
A11		A	D

Itens/ Alunos	Experiência	Identificar E/ S/ P	Portugol
A12		A	E
A13		A	D
A14		A	D
A15		B	D
A16		A	E
A17		A	E
A18	R(2)	A	E
A19		C	E
A20		A	D
A21		A	E
A22		A	E
A23		A	E
A24		A	E
A25		A	E
A26	Exp.	A	E
A27	R(2)	A	D

Quadro 11 - Análise individual da solução proposta pelos alunos para a questão 1 – 02/07

A seguir a descrição da segunda questão avaliada:

Questão 2- 02/07) O IMC – Índice de Massa Corporal é um critério da Organização Mundial de Saúde para dar uma indicação sobre a condição de peso de uma pessoa adulta. A fórmula é IMC = peso / (altura)^2. Elabore um fluxograma que tenha como entrada o peso e a altura de um adulto e mostre sua condição, segundo a tabela abaixo:

IMC em adultos	Condição
Abaixo de 18,5	Abaixo do peso
Entre 18.5 e 25	Peso normal
Entre 25 e 30	Acima do peso
Acima de 30	Obeso

A particularidade desta questão é o fato de solicitar a solução do algoritmo utilizando fluxograma, e não pseudocódigo, como nas demais. Indo ao encontro das idéias de Duval (DUVAL 1995), temos procurado trabalhar com a representação do algoritmo em diversos formatos, solicitando aos alunos que ora proponham o algoritmo em pseudocódigo, ora o façam através do fluxograma. É interessante observar o aumento significativo de alunos que representaram corretamente o encadeamento das estruturas de seleção, se comparados à questão 1 – 02/07. Ressaltamos que o raciocínio envolvido no processo de encadeamento sugerido, tanto pela questão 2, quanto pela questão 1, é similar, como mostram as figuras 2 e 3.

A Figura 3 apresenta uma opção de solução da questão 2 – 02/07, utilizando fluxograma.

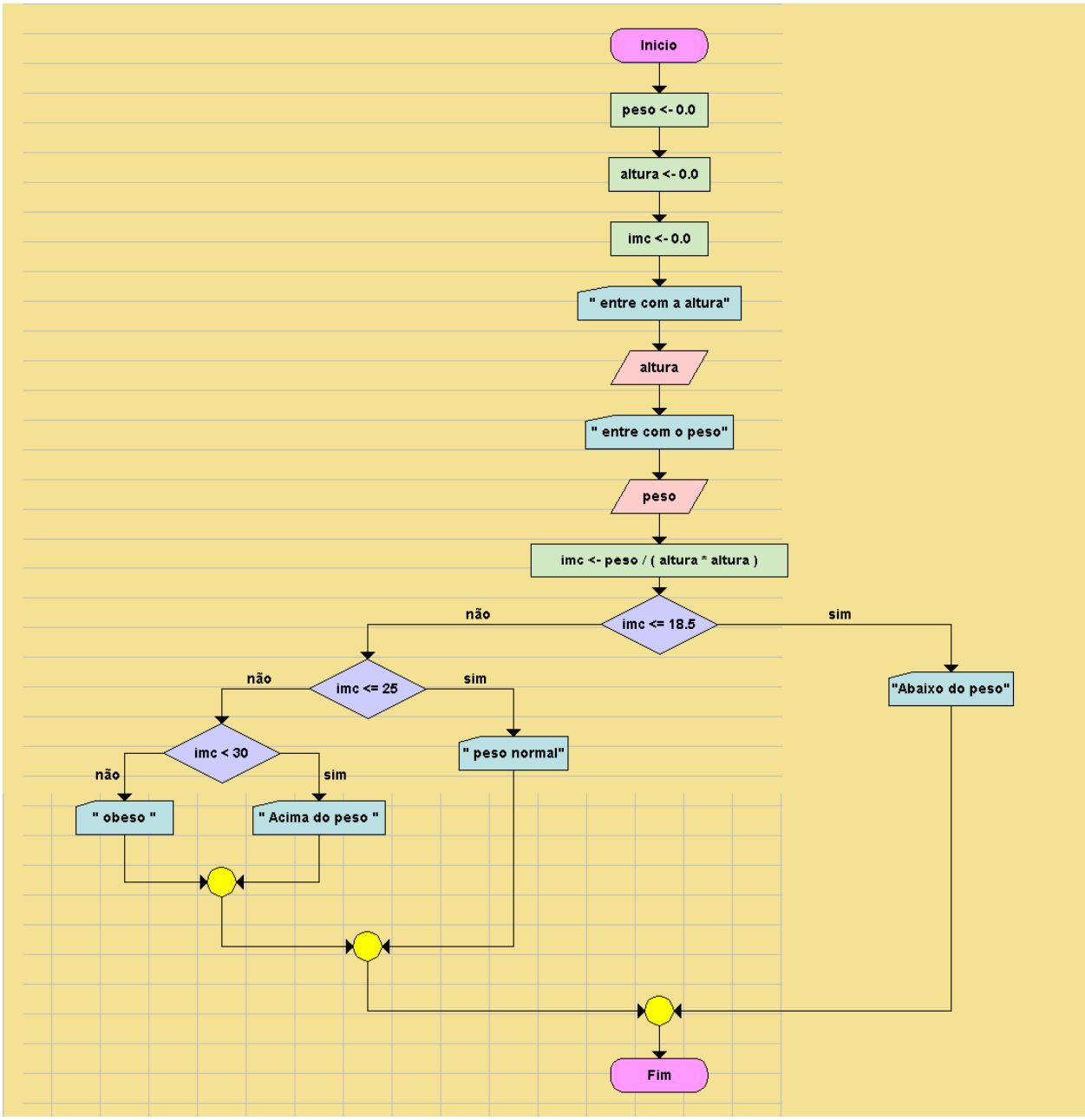


Figura 3 – Algoritmo representado em fluxograma para a questão 2 – 02/2007

Fonte: A autora.

A análise desta questão seguiu os mesmos moldes da questão anterior, inclusive as categorias utilizadas são as mesmas, porém, restritas a C, D, E e F. O Quadro 12 apresenta os dados sintetizados.

Item/ Categorias	Elaboração do Fluxograma
C: Não realizou	0
D: Sequencial	3
E: Encadeado	23
F: Incorreto	1

Quadro 12 – Síntese geral da solução proposta pelos alunos para a questão 2 – 02/07

As soluções individuais dos alunos estão ilustradas no Quadro 13.

Item/ Alunos	Experiência	Elaboração do Fluxograma
A1	R(3)	D
A2		E
A3		F
A4	R(2)	E
A5		D
A6		E
A7		E
A8		E
A9		E
A10		E
A11		E
A12		E
A13		E
A14		E
A15		E
A16		E
A17		E
A18	R(2)	E
A19		E
A20		E
A21		E

Item/ Alunos	Experiência	Elaboração do Fluxograma
A22		E
A23		D
A24		E
A25		E
A26	Esp.	E
A27	R(2)	E

Quadro 13 – Análise individual da solução proposta pelos alunos para a questão 2–02/06

Analisando o Quadro 10 – Síntese geral da solução proposta pelos alunos para a questão 1 – 02/07, observa-se que, praticamente, todos os alunos que participaram da análise tiveram sucesso para identificar, em linguagem natural, as entradas, as saídas e o modo de processamento do problema proposto na questão 1 - 02/07. No segundo item analisado, referente ao encadeamento da estrutura de seleção utilizando portugol, aproximadamente 50% dos alunos conseguiram perceber que o encadeamento era a solução mais adequada. Entretanto, outros 50% propuseram a solução de forma sequencial.

Na questão 2 – 02/07, é interessante destacar que, com a utilização do fluxograma para representar o algoritmo proposto, o número de alunos que identificaram a necessidade de utilizar o encadeamento foi muito superior ao exercício anterior, da mesma avaliação, como mostrou o Quadro 12. Isso nos leva a considerar que, nessa etapa inicial da aprendizagem, a visualização da solução, por meio de uma ferramenta gráfica, possibilita ao aluno identificar soluções com mais clareza.

Concluímos, assim, que a utilização do fluxograma deveria ser incentivada, em paralelo à linguagem textual (portugol), nesse estágio da aprendizagem.

7.3 – Elaboração do Raciocínio Matemático x Computacional

O objetivo deste caso foi analisar a relação entre a elaboração do raciocínio matemático e do raciocínio computacional pelos alunos, buscando verificar se aqueles alunos que elaboraram o raciocínio matemático de forma coerente para resolver o problema conseguem elaborar também o raciocínio computacional correspondente, na forma de um algoritmo. Com isso, conseguimos alguns indicadores para embasar nossa discussão, na

tentativa de responder à seguinte pergunta: A dificuldade dos alunos está na elaboração do raciocínio matemático ou, posteriormente, na conversão deste raciocínio para o correspondente computacional?

Note-se que, para, a partir de uma situação problema descrita em linguagem natural, chegar à solução computacional, é necessário fazer uma mudança de registro de representação multifuncional (linguagem natural) para monofuncional (formulação matemática) e, em seguida, outra mudança, tendo como registro de representação de partida a formulação matemática e de chegada, a solução em formato computacional. Dessa forma, existem duas mudanças de representação, o que indica mais uma vez a necessidade de estudar os processos de conversão que estão implícitos neste processo.

Para transformar a formulação matemática na computacional correspondente, o aluno deve discretizar o raciocínio matemático previamente elaborado, por meio da utilização de estruturas de repetição.

Apresentamos aos alunos o seguinte problema:

Questão Fibonacci) "Um fazendeiro tem um par de coelhos recém-nascidos em um ambiente fechado. Considerando que, de um modo natural, a cada mês ocorre a produção de um par e um par começa a produzir coelhos quando completa dois meses de vida. Desejamos saber quantos pares de coelhos o fazendeiro irá possuir ao final de N meses"[7]. Pede-se:

- O esboço gráfico do problema.
- A solução matemática, chegando à fórmula que define a produção de coelhos a partir do terceiro mês.
- Baseado nas suas descobertas nos itens anteriores, faça um algoritmo em Portugol para encontrar o N-ésimo termo da sequência, sendo que N deve ser digitado pelo usuário.
- Faça o teste de mesa para N = 6.

Observamos que, no momento da leitura da questão, a professora fez uma ressalva, explicando que o objetivo não era simplesmente o desenvolvimento de uma fórmula, mas,

[7] Leonardo de Pisa, filho de Guglielmo Bonacci, daí ser chamado de Fibonacci (*filius Bonacci*), apesar de ter sido um dos introdutores dos números Hindu-arábicos na Europa e principalmente, o introdutor do zero, ficou mais conhecido pela sequência que leva seu nome. Escreveu um livro denominado *Liber Abacci*, no qual propôs um problema que está nas páginas 123 e 124, denominado Problema dos pares de coelhos (*paria coniculorum*), que acaba gerando sua famosa sequência (IFUSP, 2009). Este problema é um problema clássico no ensino de introdução à programação.

sim, a observação da regularidade na composição da sequência que os pares de coelhos iriam formar. Uma vez que esta regularidade fosse percebida, os alunos deveriam propor a solução matemática, por meio de representação matemática, ou mista (linguagem natural e linguagem matemática) e o algoritmo para solucionar o problema.

Sugerimos o teste mesa com N = 6 e solicitamos o esboço gráfico da solução, para induzir o processo de raciocínio do aluno, com relação à forma como a seqüência era gerada. Uma vez que ele identificasse a regularidade da seqüência, deveria elaborar a formulação matemática. Então, era necessário discretizar o raciocínio elaborado, ou seja, especificar os passos adequados para realizar o que foi definido pela formulação matemática, por meio de um algoritmo computacional.

Este exercício foi aplicado, como nos demais casos apresentados neste trabalho, por meio de uma prova referente à primeira parte da disciplina de Lógica Aplicada. A avaliação completa pode ser visualizada no Apêndice I-3.

Nesta primeira parte da disciplina, os conteúdos abordados envolviam: comandos de entrada e saída de dados, operadores aritméticos e relacionais, estruturas sequenciais, de seleção e de repetição.

Apresentamos uma opção de solução para o problema, com o objetivo de possibilitar a discussão sobre as soluções postas pelos alunos. Em primeiro lugar, apresentamos o desenvolvimento do esquema gráfico e da formulação matemática, para, em seguida, apresentar o algoritmo computacional.

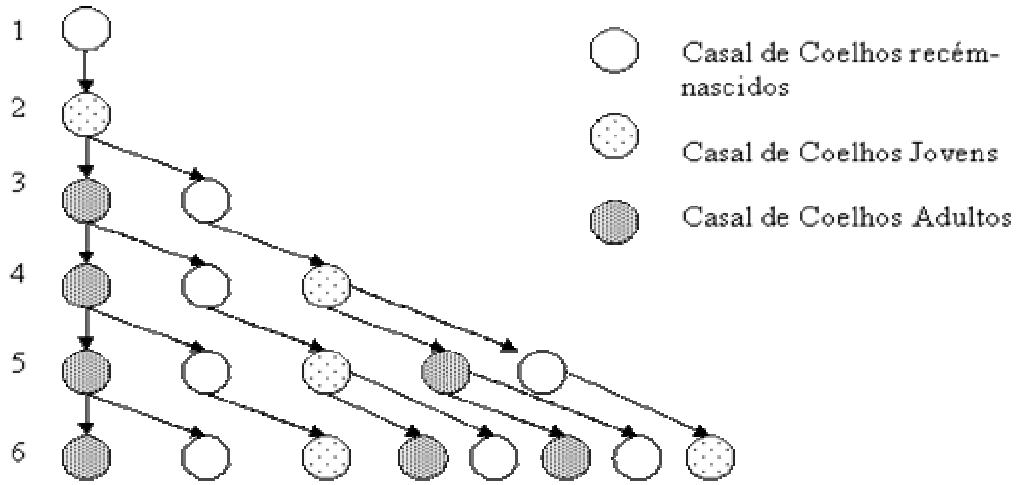


Figura 4 – Solução gráfica para o problema de Fibonacci

Fonte: A autora.

Considerando que um par de coelhos começa a produzir coelhos quando completa dois meses de vida e que o par adulto produz um par novo a cada 30 dias, temos no mês 1 apenas o par recém-nascido. No mês 2 este par se torna jovem, para, no mês 3, produzir um novo par. Então, no mês 3 existirão dois pares de coelhos: 1 par adulto + 1 par recém-nascido.

No mês 4, o par adulto produzirá de novo mais um par, enquanto que o par jovem terá completado 1 mês de vida e ainda não estará apto a produzir. Assim, no início do terceiro mês existirão três pares de coelhos, sendo: 1 par adulto + 1 par com 1 mês de idade + 1 par recém-nascido.

No mês 5, existirão dois pares adultos, sendo que cada um já produziu um novo par e um par novo que completou 1 mês, logo teremos 5 pares: 2 pares adultos + 1 par com 1 mês + 2 pares recém-nascidos.

Finalmente, no mês 6 existirão três pares adultos, sendo que cada um já produziu um novo par e dois pares novos que completaram 1 mês de vida; assim, teremos 8 pares: 3 pares adultos + 2 pares com 1 mês + 3 pares recém-nascidos.

Este processo continua até o número de meses desejado. A sequência numérica formada é conhecida como sequência de Fibonacci, e indica o número de pares ao final de cada mês: {1, 1, 2, 3, 5, 8, 13, 21, 34, ...}

Esta sequência de números tem uma característica especial denominada recursividade

e pode ser formulada pela seguinte definição:

$F_n = F_{n-2} + F_{n-1,}\ n \geq 2$

$F_1 = 1,$

$F_2 = 1.$

O que nos leva à sequência: 1 1 2 3 5 8 13 ...

$F_1\ F_2\ F_3\ F_4\ F_5\ F_6\ F_7$...

Na sequência de Fibonacci, fazendo analogia com a situação proposta no enunciado da questão, deve-se notar que o primeiro termo (F_1) corresponde ao primeiro mês, sendo o segundo termo (F_2) correspondente ao segundo mês, e assim sucessivamente.

A seguir, apresentamos a solução algorítmica em portugol para solucionar o problema proposto.

Algoritmo Fibonacci

```
inicio          // início do algoritmo em portugol
   inteiro atual, ant1, ant2, num_termos;
   escreva (("Entre com o número de meses desejado");
   leia (num_termos);
   enquanto ( num_termos ≤ 0 ) faça
      escreva (("O número de meses deve ser maior que 0");
      leia (num_termos);
   fim-enquanto;
   ant1 ← 1;
   ant2 ← 1;
   escreva (" A sequência formada será = ");
   escreva (ant2, ant1);          // escreve os 2 primeiros termos
   enquanto ( num_termos > 2 ) faça
      atual ← ant1 + ant2;
      ant2  ← ant1;
      ant1  ← atual;
      num_termos ← (num_termos – 1);
      escreva (atual);          // escreve o termo atual
```

fim-enquanto;
fim

Observa-se que o processo de discretização, ao qual estamos nos referindo, pode ser percebido na forma como a estrutura de repetição deve ser construída, para que o algoritmo gere a sequência desejada. O que acontece com alunos que apresentam dificuldades em realizar a passagem do raciocínio matemático para o correspondente computacional é que, embora eles saibam qual é o ponto de partida e o de chegada, o processo de discretização, em que devem ser especificados os passos necessários para realizar a tarefa desejada, torna-se um obstáculo, fazendo com que eles "patinem" como diria Berlinski (2002), impedindo-os de elaborar a solução adequada. O que pode ser percebido, em algumas das soluções apresentadas a seguir, notadamente nas soluções propostas pelos alunos A5, A9, A11 e A16.

Apresentamos as soluções propostas pelos alunos, nominados novamente de A1, A2 ... A27.

O aluno A1 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, porém, não elaborou uma fórmula. Implementou o algoritmo corretamente, utilizando a estrutura de repetição. A particularidade foi o fato do aluno ter utilizado vetores, o que mostra seu conhecimento prévio sobre programação, considerando que este assunto ainda não havia sido abordado.

O aluno A2 esboçou o raciocínio matemático, chegando à seguinte conclusão:

"1. O número de pais anterior é o número de filhos da próxima.

2. O número de filhos da anterior é somado com o número de pais da próxima."

Observa-se que ele não percebeu corretamente a regularidade entre os elementos da sequência, não chegando à forma como ela era gerada. Elaborou apenas um esqueleto do algoritmo, com a declaração de algumas variáveis e um comando de repetição incompleto.

O aluno A3 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, elaborou a formulação em linguagem natural: "O número de coelhos do mês escolhido será sempre igual à soma do número de coelhos dos 2 meses anteriores". Implementou o algoritmo corretamente, fazendo uso adequado do comando de repetição.

O aluno A4 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, elaborou a fórmula "T = M[-1]+M[-2]".

Implementou o algoritmo corretamente, fazendo uso adequado do comando de repetição. A particularidade estava na experiência autodidata do aluno com programação.

O aluno A5 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo a sequência gerada, conseguiu perceber a regularidade da seqüência, mas não elaborou a formulação adequadamente. Depois do esquema gráfico, elaborou o seguinte raciocínio:

"Número de coelhos adultos x 2 + coelhos filhotes

solução matemática à fórmula: "cA x 2 + cF

//coelhos do mês passado + coelhos do mês retrasado= número de coelhos desse mês".

Percebe-se que o aluno teve dificuldade em representar, por meio da linguagem matemática, o que formulou em linguagem natural, e, como elaborou o algoritmo baseado nesta formulação, o fez incorretamente. Não realizou a discretização do raciocínio, necessária para a elaboração de um algoritmo computacional, apenas aplicou a fórmula descrita anteriormente, da seguinte forma:

```
"inicio          // início do algoritmo em portugol
    inteiro (n, a, cA, cF)
    enquanto (a ≠ 0);
    escreva ("Quantos meses ?") = n;
    leia n;
    a = n-1;
    coelhos = (coelhos de n-1) + (coelhos de n-2)
fim."
```

O aluno A6 elaborou o raciocínio matemático por meio de um esquema gráfico, porém, do mês 5 para o mês 6, cometeu um erro o que o levou à sequência: 1 1 2 3 5 7 11. Assim, não conseguiu formular a solução corretamente, não chegou a elaborar uma fórmula e implementou o algoritmo incorretamente.

O aluno A7 elaborou o raciocínio matemático por meio de um esquema gráfico, porém, do mês 6 para o mês 7 cometeu um erro, da mesma forma que o aluno A6, que o levou à sequência: 1 1 2 3 5 8 15. Assim, não conseguiu formular uma solução, não chegou a elaborar uma fórmula e não implementou o algoritmo.

O aluno A8 elaborou o raciocínio matemático corretamente, por meio de um esquema

gráfico, descobrindo a sequência gerada até o mês 5, entretanto, não conseguiu obter a regularidade corretamente, consequentemente chegou à seguinte formulação em linguagem natural, “Número do Coelhos = Soma de pares de coelhos do mês anterior, coelhos férteis e não férteis”. Não implementou o algoritmo.

O aluno A9 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, elaborou a fórmula corretamente “$N_x = N_{x-1} + N_{x-2}$”. Implementou o algoritmo incorretamente, porque não conseguiu realizar a discretização, para transformar o raciocínio matemático no correspondente computacional. A seguir, o algoritmo proposto pelo aluno:

```
“inicio
// declaração das variáveis
   real: mes, n1, cont, n,;
   n1 ← 1;    n ← 0;    cont  ← 0;
   escreva (" Digite o número de meses ");
    leia (mes);
   enquanto ( cont < mes ) faça      //A sintaxe do comando de repetição está correta
      nl ← n1 + n-1;                 // Porém não realiza o que o aluno gostaria de acordo
      n ← n1 + n;                    // com a formulação previamente elaborada por ele.
      cont ← (cont + 1);
   fim-enquanto;
fimalgoritmo.”
```

O aluno A10 elaborou o raciocínio matemático corretamente, por meio de um esquema numérico, descobrindo como a sequência era gerada. Reconheceu que se tratava de sequência de Fibonacci. Não elaborou uma fórmula e implementou o algoritmo corretamente, utilizando estrutura de repetição.

O aluno A11 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, elaborou a fórmula corretamente “$N_x = N_{x-1} + N_{x-2}$”, entretanto, implementou o algoritmo incorretamente; porque não conseguiu realizar a discretização, para transformar o raciocínio matemático no correspondente computacional, como pode ser percebido no algoritmo proposto pelo aluno.

```
“inicio
   inteiro mes, na, nb, nc, total;
```

```
   escreva (" Entre com o mês ");
   leia (mes);
   enquanto ( n ≤ mes  ) faça        //A sintaxe do comando de repetição está correta,
      na ← 1;                        // Porém ele inicializou as variáveis dentro do laço
      nb ← 1;
      nc ← na + nb;                  // O laço não realiza o que o aluno gostaria, de acordo
      nc ← nc +nb;                   // com a formulação elaborada por ele.
      n ← n + 1;
   fim-enquanto;
   escreva ("O valor total é", total);   // a variável total não foi utilizada, está com 0.
fim."
```

O aluno A12 não resolveu a questão.

O aluno A13 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo a sequência, mas não conseguiu perceber como a ela era gerada, elaborou a fórmula "F(x) = x + 2y (?)", o que mostra que ele estava em dúvida. Não elaborou o algoritmo.

O aluno A14 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, chegou à fórmula N = (N-1) + (N-2) e implementou o algoritmo corretamente, porém de forma recursiva. A particularidade deste aluno é que ele já havia cursado parcialmente disciplina similar em outra universidade, o que explica a utilização do conceito de recursividade, embora o mesmo não tivesse sido abordado.

O aluno A15 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, elaborou a fórmula "Xn = Xn-1 + Xn-2". Implementou o algoritmo corretamente, utilizando a estrutura de repetição adequadamente.

O aluno A16, reconheceu o problema de Fibonacci, elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico e descobriu como a sequência era gerada. Não elaborou uma fórmula matemática, mas escreveu em linguagem natural: "O próximo termo é a soma dos dois últimos". Implementou o algoritmo, porém, incorretamente; porque não conseguiu realizar a discretização, para transformar o raciocínio matemático no correspondente computacional. Segue o algoritmo proposto:

"inicio

```
   inteiro numpar;                    // Não declarou todas as variáveis
    Numpar = 1; anterior = 0; N = 1;
   leia (meses);                      //A sintaxe do comando de repetição está correta
   enquanto ( n ≤ meses  ) inicio     // apenas esqueceu do faça
      anterior = numpar;
      numpar = numpar + x;            // O laço não realiza o que o aluno gostaria, de acordo
      n ← n + 1;                      // com a formulação elaborada por ele.
   fim-enquanto;
   escreva (numpar);
fim.”
```

O aluno A17 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, porém, gerou a sequência até o elemento 5, e elaborou a formulação matemática incorretamente, da seguinte forma: “N x (N-2)/N”. Fez o algoritmo incorretamente, aplicando a fórmula diretamente.

O aluno A18 não fez esquema gráfico, nem formulação matemática. Implementou o algoritmo incorretamente, da seguinte forma:

```
“inicio
   real coelho, meses;
   coelho ← 0;
   leia (meses);
   escreva (“No. total de meses =”, meses);
   coelho ← (meses) x 2;
   imprima (coelhos);          // trocou o comando escreva pelo imprima.
fim.”
```

O aluno A19 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, elaborou a seguinte formulação em linguagem natural: “Número de pares = soma de números dos últimos 2 meses”. Não fez o algoritmo.

O aluno A20 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, elaborou a fórmula “X = (X-2) + (X-1)”. Implementou o algoritmo corretamente, utilizando a estrutura de repetição adequadamente.

O aluno A21 reconheceu o problema, pois já havia cursado parcialmente disciplina similar em outra universidade. Não fez esquema gráfico, elaborou direto a fórmula “Pm = P(m-1) + P(m-2)” e implementou o algoritmo corretamente, utilizando a estrutura de repetição adequadamente.

O aluno A22 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, porém, gerou a sequência até o elemento 5, e formulou “Nt = Nc(n-1) + Nf”, o que mostra a dificuldade em perceber a regularidade da sequência. Não elaborou o algoritmo.

A aluno A23 não realizou o exercício.

O aluno A24 não elaborou esquema gráfico, partiu para a formulação matemática, propondo:

“N1= 2; N2= 4 N3= 8;

$Y = 2^n$, onde Y é a produção de coelhos.”

O aluno não obteve a sequência correta, portanto, não conseguiu obter a forma como ela era gerada. Implementou o algoritmo, aplicando a formulação obtida anteriormente, incorretamente, da seguinte forma:

“inicio

inteiro mes;

inteiro produção;

produção = 0;

escreva (“Informe o mês desejado”);

leia (mes);

produção = 2^{mes};

escreva (“A produção de coelhos no mês”, mês, era de “, produção);

fim.”

O aluno A25 elaborou o esquema gráfico incorretamente, chegando à sequência 1 1 2 3 4. Elaborou a seguinte formulação matemática:

“N. mês - 2= coelhos adultos; N. mês -1= total de pares”.

Elaborou o algoritmo incorretamente.

O aluno A26, reconheceu o problema de Fibonacci, elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico e descobriu como a sequência era gerada. Descreveu como a sequência de Fibonacci é gerada em linguagem natural: “Na sequência de

Fibonacci o termo subsequente é dado pela soma dos precedentes, assumindo que comece em 1, 1+0 =1, daí 1+1=2, 2+1=3, 2+3=5 e, assim, sucessivamente". Implementou o algoritmo corretamente, utilizando a estrutura de repetição adequadamente; a particularidade foi o fato de o aluno ter utilizado vetores, o que mostra seu conhecimento prévio sobre programação, considerando que este assunto ainda não havia sido abordado.

O aluno A27 elaborou o raciocínio matemático corretamente, por meio de um esquema gráfico, descobrindo como a sequência era gerada, não elaborou uma fórmula. Implementou o algoritmo corretamente, utilizando a estrutura de repetição adequadamente.

O Quadro 14 apresenta a síntese dos dados coletados, da mesma forma como apresentado nas seções 6.4 e 7.1. A diferença deste quadro está na disposição das informações, considerando que as soluções individuais foram discutidas anteriormente, optou-se por apresentar os itens analisados e para cada aluno se ele teve sucesso ou não, em cada item. Os itens analisados foram os seguintes:

- Experiência (Exp): indica se o aluno já havia cursado a disciplina de *Lógica de Programação* ou similar, ou se trabalhava na área de programação.
- Regularidade: mostra se o aluno conseguiu descobrir a regularidade da sequência, ou seja, como a sequência era gerada. Este dado é relevante, porque muitos alunos conseguiram chegar aos números da sequência, mas não conseguiram obter a forma como estes números eram gerados.
- Raciocínio: indica se o aluno elaborou o raciocínio matemático, ou seja, se chegou à sequência de Fibonacci, de forma gráfica, textual (utilizando a linguagem natural), ou de forma incorreta.
- Formulação: indica se o aluno formulou uma solução para o problema, utilizando linguagem matemática, linguagem natural ou de forma incorreta.
- Algoritmo: Na elaboração do algoritmo, foram encontradas soluções do tipo:
    1. iterativas: utilizando a estrutura de repetição;
    2. recursiva: utilizando a recursividade;
    3. incompleta: o algoritmo foi feito parcialmente, em muitos casos apenas um esboço;
    4. não: não implementaram uma solução algorítmica.
- Os alunos que não resolveram a questão estão indicados com – (traço) em todos os

itens.

Quanto ao preenchimento do Quadro 14 (*X)* indica que o aluno realizou adequadamente e (*–)* não realizou.

Aluno	Exp.	Regul.	Raciocínio			Formulação			Algoritmo			
			Graf.	Tex.	Inc.	Mat.	Tex.	Inc.	Ite.	Rec.	Inc.	Não
A1	X	X	X			-	-	-	X			
A2		-	-	-	-			X			X	
A3		X	X				X		X			
A4		X	X			X			X			
A5		X	X				X				X	
A6		-			X	-	-	-			X	
A7		-			X	-	-	-				X
A8		-	X					X				X
A9		X	X			X					X	
A10	X	X	X			-	-	-	X			
A11		X	X			X					X	
A12		-	-	-	-	-	-	-	-	-	-	-
A13		-	X					X				X
A14	X	X	X			X				X		
A15		X	X			X			X			
A16	X	X	X				X				X	
A17		-	X					X			X	
A18		-	-	-	-	-	-	-			X	
A19		X	X				X					X
A20		X	X			X			X			
A21	X	X	-	-	-	X			X			

Aluno	Exp.	Regul.	Raciocínio			Formulação			Algoritmo			
			Graf.	Tex.	Inc.	Mat.	Tex.	Inc.	Ite.	Rec.	Inc.	Não
A22		-	X					X				X
A23	-	-	-	-	-	-	-	-	-	-	-	-
A24		-	-	-	-			X			X	
A25		-			X			X			X	
A26	X	X	X			X	X		X			
A27		X	X			-	-	-	X			

Quadro 14 – Análise individual da solução proposta pelos alunos para a questão Fibonacci

Estes resultados estão sintetizamos no Quadro 15, de acordo com o raciocínio desenvolvido pelos alunos. O número de alunos foi totalizado para mostrar com mais clareza a proporção de alunos em relação ao raciocínio elaborado. Cada forma de raciocínio foi classificada como uma categoria, descrita a seguir.

A: O aluno chegou ao resultado da forma esperada, ou seja, elaborou o raciocínio, percebeu a regularidade da sequência e implementou o algoritmo corretamente, entretanto, o aluno possuía algum tipo de experiência com o desenvolvimento de algoritmos.

A': O aluno chegou ao resultado da forma esperada, ou seja, elaborou o raciocínio, percebeu a regularidade da sequência e implementou o algoritmo corretamente. Neste caso, o aluno não possuía experiência com o desenvolvimento de algoritmos.

B: O aluno elaborou o raciocínio, percebeu a regularidade da sequência, mas não conseguiu implementar o algoritmo corretamente, ou não elaborou o algoritmo. Ou seja, não conseguiu discretizar o raciocínio matemático de forma a transformá-lo no correspondente computacional.

C: O aluno elaborou o raciocínio, chegou à sequência, mas não percebeu a regularidade; logo, implementou o algoritmo incorretamente ou não implementou.

D: O aluno esboçou uma formulação incorreta e implementou o algoritmo incorretamente ou não implementou.

E: Não realizou o exercício.

Categoria	A	A’	B	C	D	E
No. Alunos	5	5	4	5	6	2
Raciocínio Matemático (regularidade)	51,8%			48,2%		
	37%		14,8%	18,5%	22,2%	7,4%
Raciocínio Computacional	37%		63%			

Quadro 15 – Síntese das soluções propostas pelos alunos para a questão Fibonacci

Face ao estudo de caso apresentado, observa-se que, aproximadamente, 51% dos alunos resolveram o problema matemático, ou seja, construíram a sequência de forma correta e perceberam a regularidade presente no processo de construção da mesma, o que corresponde às categorias A, A’e B.

Entre os 51% de alunos, 37% obtiveram sucesso na elaboração da solução computacional (algoritmo), embora a metade deles possuísse algum tipo de experiência com programação. Deve-se considerar que alguns alunos, que cursaram disciplinas similares, o fizeram parcialmente, justamente porque ao se sentirem incapazes de compreender o raciocínio computacional, do qual estamos falando ao longo deste trabalho, abandonaram a disciplina prematuramente.

Com relação à categoria B, que corresponde a 14,8% dos alunos, embora eles tenham elaborado a solução matemática corretamente, não conseguiram propor a solução correspondente no formato computacional, o que evidencia a dificuldade na passagem do raciocínio matemático para o computacional, dificuldade esta, ligada ao processo de discretização necessário para transformar o raciocínio matemático no correspondente computacional.

Comparando-se as soluções propostas pelos quatro alunos, que compõem esta categoria (A5, A9, A11 e A16), observa-se que três deles, a saber, A9, A11 e A16 não conseguiram elaborar uma seqüência de comandos, dentro da estrutura de repetição, que executasse os passos necessários para realizar o que à formulação matemática, proposta por

eles, indicava. O aluno A5 foi o único, dentre os quatro, que apresentou problemas também relacionados à sintaxe da estrutura de repetição.

A dificuldade em discretizar o raciocínio matemático, para conceber o raciocínio computacional, aliada à dificuldade em perceber a regularidade das situações em análise, observada na categoria C, correspondente a 18,5% do total de alunos, constituem a principal barreira na aprendizagem de algoritmos.

Observou-se que os alunos que não elaboraram o raciocínio matemático adequadamente, na maioria dos casos não propuseram a solução algorítmica, porém, alguns chegaram a um esboço. Tendo em vista o estágio embrionário destes esboços, para efeito de nossas conclusões, consideramos as categorias D e E como sendo os alunos com maiores dificuldades de aprendizagem, tanto em nível matemático quanto computacional. Estas categorias representaram aproximadamente 30% do total de alunos, o que mostra a deficiência destes alunos em resolver problemas, o que vai ao encontro das considerações apontadas por Friedmann (2005).

Analisando as categorias B, C, D e E, observamos que os alunos, que não conseguiram propor uma solução algorítmica para o problema apresentado, somam aproximadamente 63%, o que vem a confirmar a relevância deste e de outros trabalhos que buscam compreender as formas de raciocínio envolvidas neste processo para subsidiar técnicas e metodologias de ensino que promovam a aprendizagem deste conteúdo.

CAPÍTULO 8

Conclusões e Considerações Finais

Faz necessário, um breve relato, sobre uma jovem face à escolha de um curso superior. Esta jovem, com grande afinidade com a Matemática, dirigiu-se a professora responsável por tal disciplina a fim de obter informações sobre o curso de Ciência da Computação.

Ao ser questionada, sobre quais dificuldades poderiam ser encontradas nesse curso, a professora fez o seguinte comentário: “Quanto às disciplinas de matemática (Cálculo I, Cálculo II, Cálculo III, Geometria Analítica, etc...) não deve haver problemas, entretanto, existe uma matemática aplicada à computação, que exige um raciocínio próprio... Costuma-se dizer que ou a pessoa tem aptidão, ou não”.

Durante o primeiro mês da disciplina de *Fundamentos da Programação*, cujo conteúdo central era a elaboração de algoritmos, a jovem percebeu que tratava-se daquela matemática aplicada à computação.

Embora este breve relato não esteja diretamente relacionado a este trabalho, ele é oportuno, pois explica a origem das nossas inquietações. Neste trabalho, associamos os termos ‘pensamento algorítmico’ ou ‘pensamento computacional’ para nos referir àquela matemática, própria da computação.

A educação, em geral, passou por muitas transformações e as dificuldades dos alunos deixaram de ser encaradas como mera falta de aptidão. Diversas pesquisas têm sido realizadas com o objetivo de promover a aprendizagem nas mais diversas áreas. O capítulo 5 deste trabalho, apresenta algumas destas pesquisas, por terem papel relevante para o ensino de algoritmos, ou introdução à programação.

O diferencial entre estas pesquisas e o trabalho que estamos finalizando é que nosso enfoque esteve na concepção e no desenvolvimento do raciocínio computacional, enquanto que o delas estava no desenvolvimento de ferramentas para o apoio à aprendizagem. Para compreender este processo, foi necessário realizar uma análise do conceito de algoritmo, que resultou em indícios de obstáculos epistemológicos. Aliado a estes indícios, detectamos a necessidade de avaliar o fenômeno da não-congruência, de acordo com Duval (1995), na passagem de uma representação em linguagem natural, ou algébrica, para a correspondente em linguagem computacional.

Buscamos em meio a inúmeras possibilidades, um referencial teórico que desse suporte às nossas necessidades. Embora o conceito de ‘obstáculo epistemológico’ seja normalmente associado ao desenvolvimento histórico de um conceito, optamos por adaptá-lo ao nosso problema, uma vez que o objeto de estudo estava ligado à forma de compreender um conhecimento.

Considerando os resultados dos trabalhos analisados, aliados à nossa experiência docente, tínhamos como hipóteses que a estrutura de repetição fosse um obstáculo à passagem do raciocínio matemático para o correspondente computacional. Com o objetivo de buscar subsídios para avaliar esta hipótese, realizamos o primeiro estudo de caso descrito na seção 7.1. As conclusões deste estudo reforçaram esta hipótese, porém motivaram outros questionamentos, tais como:

1. As formas de representação, utilizadas no ensino de algoritmos, promovem a conversão de registros de representação semiótica, necessária à passagem do raciocínio matemático ao correspondente computacional?
2. Os alunos, capazes de resolver problemas matemáticos, também são capazes de resolver problemas computacionais?
3. A dificuldade dos alunos está em elaborar o raciocínio matemático, necessário para conceber a solução matemática, ou está no processo de discretização envolvido na transformação do raciocínio matemático no correspondente computacional?

Frente a estes questionamentos, fez-se necessário uma investigação mais profunda sobre os registros de representação semiótica mobilizados para representar o pensamento computacional. Concluímos, então, que o fenômeno da não-congruência presente na passagem das unidades significantes no registro de partida em linguagem natural, ou algébrica, para as unidades significantes no registro de chegada, em linguagem computacional, são mais um fator produtor de obstáculos à aprendizagem. Além disso, fizemos uma análise dos tipos de representação mais utilizados para o ensino de algoritmos e concluímos que, embora existam diversas formas de representação, como as citadas na seção 4.5, o pseudocódigo é priorizado, tendo as demais, papel secundário.

Com este cenário, realizamos algumas atividades em três turmas de introdução à programação, do Curso de Tecnologia em Informática, da UTFPR, que pudessem dar subsídios para avaliar nossas hipóteses e responder a alguns de nossos questionamentos. As

atividades foram selecionadas e deram origem aos estudos de casos aqui apresentados, o que possibilitou a análise do desempenho dos alunos com tipos de registros de representação semiótica diferentes e a relação entre a habilidade em resolver problemas envolvendo raciocínio matemático e raciocínio computacional.

O estudo de caso, relativo às formas de representação, descrito na seção 7.1, revelou que os alunos elaboravam o raciocínio esperado mais facilmente, utilizando fluxograma, se comparado com o pseudocódigo. Apesar de ambos estarem classificados, segundo Duval (1995), como registros monofuncionais, por meio do fluxograma os alunos perceberam mais claramente a relação hierárquica entre as ações. Por ser uma representação gráfica, ele se mostrou mais intuitivo e adequado para a introdução à aprendizagem de algoritmos.

Visando estabelecer uma relação entre a habilidade do aluno em elaborar um raciocínio matemático e em elaborar o correspondente raciocínio computacional, na forma de um algoritmo, propusemos a atividade descrita na seção 7.2, na qual constatamos que pouco mais da metade dos alunos tiveram êxito na solução do problema matemático. Desses alunos, 71% foram capazes de também solucionar o problema no formato computacional.

Este estudo de caso nos levou à conclusão que os alunos, mesmo aqueles que possuem habilidades em resolver problemas matemáticos, enfrentam dificuldades em conceber o raciocínio computacional. Isto se deve ao processo de discretização, necessário à transformação do raciocínio matemático no correspondente computacional. Analisando as soluções propostas pelos alunos A5, A9, A11 e A16, concluímos que eles não conseguiram superar o obstáculo, inerente a este processo.

Outra conclusão, oriunda do estudo de caso apresentado na seção 7.2, foi que: a dificuldade em discretizar o raciocínio matemático, para conceber o raciocínio computacional, aliada à dificuldade em perceber a regularidade das situações em análise, constituem a principal barreira na aprendizagem de algoritmos.

Por outro lado, os alunos que não elaboraram o raciocínio matemático adequadamente, na maioria dos casos não propuseram a solução algorítmica, e aqueles que esboçaram algum tipo de solução não tiveram êxito, como discutido na seção 7.2. O que nos levou a concluir que, a capacidade de elaborar o raciocínio matemático adequadamente, não é suficiente para garantir o sucesso na elaboração do raciocínio algorítmico correspondente, mas é necessário.

Um dado preocupante, obtido com este estudo de caso, foi que mais de 60% dos

alunos não conseguiram propor uma solução algorítmica para o problema e, aproximadamente, 30% também não conseguiram elaborar uma solução matemática, o que mostra o despreparo destes alunos em relação à resolução de problemas, embora tenham passado por um rigoroso processo de seleção, considerando-se a relação de aproximadamente dez candidatos por vaga.

Como apontado por Friedmann (2005), se o conteúdo referente à Matemática, abordado no ensino fundamental e médio, incluísse tópicos ligados à matemática discreta, os alunos poderiam desenvolver as habilidades de abstração e discretização necessárias à concepção do raciocínio computacional.

Finalmente, ressaltamos a relevância de trabalhos que busquem subsídios epistemológicos para fundamentar técnicas e metodologias de ensino que promovam os processos de aprendizagem.

8.1– Trabalhos Futuros

Sabe-se que a conclusão de uma tese é o ponto de partida para uma série de outras pesquisas. Neste caso, em primeiro lugar, acreditamos que é necessário aprofundar as pesquisas sobre o conteúdo da disciplina de Matemática no ensino fundamental e médio, assim como realizado por Friedmann (2005), pois o desenvolvimento do raciocínio matemático, principalmente ligado à matemática discreta, poderá trazer benefícios no desenvolvimento do raciocínio computacional.

Faz-se necessário, também, analisar a aprendizagem da estrutura de repetição por meio de outras formas de representação, notadamente fluxogramas, pois a melhoria alcançada nas estruturas condicionais aninhadas é um indício favorável à sua utilização.

Utilizamos algumas ferramentas para o ensino de algoritmos, como o *Portugol IDE e o Visualg*, e concluímos que as mesmas tiveram um impacto positivo, estimulando a aprendizagem. Porém, a avaliação mais rigorosa da utilização das mesmas fugiu ao escopo deste trabalho. Consideramos que uma pesquisa nesta direção poderia ser frutífera.

Baseado nos obstáculos identificados, propor uma metodologia de ensino que possa minimizá-los, por meio da utilização de fluxogramas como representação intermediária entre a linguagem natural e a linguagem computacional, visando abrandar o fenômeno da não-congruência observado entre elas.

Observa-se que os avanços tecnológicos deverão ter um grande impacto nas formas de raciocínio computacional, pois a elaboração de algoritmos está diretamente ligada à tecnologia utilizada na construção dos computadores. Uma vez que esta tecnologia tenha um novo formato, diferente do sistema binário utilizado pelo computador digital, poderá ser necessário realizar uma nova análise dos processos envolvidos na transformação do raciocínio neste novo formato.

Referências

ABELSON, H.; ABELSON, A. *Logo for Macintosh. An Introduction Through Object Logo*. MIT Press Classics Series, 1993.

ALIBERT, D.; THOMAS, M. Research on Mathematical Proof. In D. Tall (Ed.), *Advanced Mathematical Thinking* (pp. 215-230). Dordrecht, The Netherlands: Kluwer Academic Publishers, 1991..

ARTIGUE, M. (1988). *Ingéniérie Didactique*. Reserches en Didatique des Mathématiques. Grenoble, Vol.10, nº 3, p. 281-308, 1988.

ARTIGUE, M. (1990). *Epistémologie et Didactique*. Reserches en Didatique des Mathématiques. Vol. 10, Nº 23, 1990.

ARZARELLO, F.; BAZZINI, L.; CHIAPPINI, G. *L'Algebra como strumento di pensiero. Analisi teorica e considerazioni didattiche.* Progetto Strategico CNR – TID, Quaderno n. 6, 1994.

ASCENCIO, A. F.; CAMPOS, E.A. *Fundamentos da programação de Computadores*. São Paulo: Prentice Hall, 2002.

BACHELARD, G. *A formação do Espírito Científico*. São Paulo: Contra-ponto, 1996.

BARBOSA, L.M. *Ensino de Algoritmos em Cursos de Computação.* Dissertação de Mestrado. São Paulo: EDUC, 2001.

BEN ARI, M. Constructivism in Computer Science Education. Journal of Computers In: *Mathematics and Science Teaching*, Vol. 20, 2001.

BERLINSKI, D. *O advento do Algoritmo: A Idéia que Governa o Mundo*. São Paulo: Ed. Globo, 2002.

BOMBELLI, R. *L'Algebra*. Con la introducción de U. Forti y el prefacio y el análisis de E. Bortolotti. (Feltrinelli: Milano), 1966.

BOUD, D.; FELETTI, G. *The challenge of problem-based learning* (2. ed.). London: Kogan Page, 1997.

BROLEZZI, A. C. *A Tensão entre o Discreto e o Contínuo na História da Matemática e no Ensino da Matemática.* Tese de doutorado. Faculdade de Educação – USP, 1996.

BROUSSEAU, G. *Los Obstáculos Epistemológicos y los Problemas en Matemáticas.* Recherches en Didactique des Mathématiques, Vol. 4 (2), 1983. Disponível em:

http://lem.usach.cl/biblioteca/index.php?Tipo=4&Tema=7. Acessado pela última vez em 05/05/2008.

BROUSSEAU, G. *Fondements et méthodes de la didactique des mathématiques.* Recherche en Didactique des Mathématiques. Grenoble, Vol.7, nº 2, p. 33-115, 1986.

BROUSSEAU, G. *Le contrat didactique: le milieu.* Reserches en Didatique des Mathématiques. Grenoble, Vol. 9, nº 3, p. 309-336, 1988.

BROUSSEAU, G. *Theory of didactical situations in mathematics.* Dordrecht, The Netherlands: Kluwer. (Edited and translated by N. Balacheff, M. Cooper, R. Sutherland, and V. Warfield), 1997.

BRUYNE, P.; *Dinâmica da Pesquisa em Ciências Sociais: os Pólos da Prática Metodológica.* Rio de Janeiro, F. Alves, 1991.

CHEVALLARD, Y. *La transposición didáctica:del saber sabio al saber enseñado.* Buenos Aires: Aique. 1991.

CHI, M. T. H.; GLASER, R.; FARR, M. (Eds.). *The Nature of Expertise*. Hillsdale, Lawrence Erlbaum Associates, 1988.

CHURCH, A. *A note on the Entscheidungsproble*. The Journal of Symbolic Logic, Vol. 1, No. 1. p. 40-41, 1936.

CORNU, B. *Limits*. *Advanced Mathematical Thinking (cap. 10)*. David Tall, Kluwer Acadmic Publishers. Dordrechet/Boston/London 1991.

CORREIA, A. M. A.; CHENG, L. Estudantes e Professores de Perfil. In: XXIX CONGRESSO BRASILEIRO DE ENSINO DE ENGENHARIA, p.197- 205. Porto Alegre, RS, 2001. Disponível em: http://www.pp.ufu.br/Cobenge2001/titulos.pdf . Acessado pela última vez em 07/12/2008.

COSTELLOE, E. *Teaching Programming The State of the Art.* Department of Computing, Institute of Technology Tallaght, Dublin 24. CRITE Technical Report, 2004a.

COSTELLOE, E. *The Use of a Software Enabled Scaffolding Environment to Aid Novice Programers.* Submited to the University of Dublin for the Degree of Master of Science, 2004b.

DEDEKIND, R. Was sind und was sollen die Zahlen? In: *Gesammelte Mathematische Werke*. New York: Chelsea Publishing Company. Vol. 3, p. 335-391, 1969.

DOUROX, A. *La valeur absolue: difficultes majeurs pou une notion meneure*. Petit (3),1983.

DUVAL, R. *Semiosis y Pensamiento Humano – Registros Semióticos y Aprendizagemns Intelectuais*. Universidade Del Valle, Colômbia. 2004. Tradução do original: DUVAL, R. *Sémiosis et pensée humaine – registres sémiotiques et apprentissagens intellectuels*. Berne, Peter Lang, 1995.

DUVAL, R. Registros de representações semióticas e funcionamento cognitivo da compreensão em Matemática. In: *Aprendizagem em Matemática*. Machado, S. D. A. (org.). p. 11-33. Campinas, SP: Papirus, 2003.

ERDTMANN, M. *Princípio Básico de Computação e Linguagem*. Material didático apresentado na disciplina S2I (Sistemas Industriais Inteligentes). UFSC, 2007.

ERNEST, P. *Mathematical Knowledge and Context, Situated Cognition and the Learning of Mathematics* (Anne Watson, Ed.), Oxford: University of Oxford Department of Educational Studies. Chapter 1, p. 13-29, 1998.

FARRER, H. *Algoritmos Estruturados.* São Paulo: LTC, 1999.

FEKETE, A.; KAY, J.; KINGSTON, J.; WIMALARATNE, K. *Supporting Reflection in Introductory Computer Science.* Anais do 31º ACM SIGCSE (Technical Symposium on Computer Science Education), Vol. 32, 2000.

FELDER, R. M.; SILVERMAN, L.K. *Learning and Teaching Styles in Engineering Education*. Engineering Education, Vol. 78, Nº7, 1988.

FORBELLONE, A. L.; EBERSPACHER, H.F. *Lógica de Programação*. São Paulo: Makron Books, 2000.

FRIEDMANN, C. V. P.; JURKIEWIZ, S.; LOZANO, A. *Algumas questões sobre algoritmos, modelagem discreta e ensino.* Anais da IV Conferência Nacional sobre Modelagem e Educação Matemática, Feira de Santana, 2005.

FRIEDMANN, C.V.P. *Matemática Discreta, Algoritmos, Modelos. Tendências do Ensino de Matemática no Início do Século XXI.* Tese de doutorado apresentada ao programa de Engenharia de Produção-UFRJ, Brasil, 2003.

GAMA, M.C. A *Teoria das Inteligências Múltiplas e suas implicações para Educação.* Trait Tecnologia Ltda, 1998.

GARDNER, H. *Frames of mind.* New York, Basic Books Inc., 1985.

GARDNER, H. *The mind's new science.* New York, Basic Books Inc., 1987.

GARDNER, H. *Frames of Mind: The theory of multiple intelligences*. New York: Basic

Books, 1993.

GERSTING, J. L. *Fundamentos Matemáticos da Computação.* São Paulo: LTC, 1995.

GLAESER, G. *Épistémologie des nombres relatifs.* RDM, V. 2, n.3 p. 303-346. Grenoble, 1981.

GÖDEL, K. *Collected Works.* Editado por Solomon Feferman e outros. Oxford: Oxford University Press, 2v. 1986-1990.

GUIMARÃES, M. A. M. *Um Ambiente para Ensino de Algoritmos Introdutórios.* Tese de doutorado, apresentada ao Departamento de Informática da PUC/RJ, 1995.

HARRIS, J. *CS1: Where is Visual Basic?* Consortium for Computing Sciences in Colleges. Proceedings of the fourteenth annual consortium on Small Colleges Southeastern conference, EUA, 2000.

HENDERSON, P. B. *Anatomy of an introductory Computer Science Course*. Anais do 17º ACM SIGCSE (Technical Symposium on Computer Science Education), 1986.

HENDERSON, P. B. *Modern Introductory Computer Science*. Anais do 18º ACM SIGCSE (Technical Symposium on Computer Science Education), 1987.

HUMEL, J.; MEHTA, J. *Using Visual Basic in the CS Curriculum.* ACM SIGCSE, Vol.34, 2002.

HUNDHAUSEN, C. *Integrating Algorithm Visualization Technology Into an Undergraduate Algorithms Course: Ethnographic Studies of a Social Constructivist Approach*. Computers & Education. Vol. 39, Issue 3, p. 237 – 260, 2002.

JENKINS, T. *Teaching Programming – A Journey from Teacher to Motivator*. First LTSN-ICS Conference on Teaching Programming, 2001.

ISRAEL, D. *Reflections on Gödel's and Gandy's Reflection on Turing's Thesis*. Minds and Machines, N.12 , p. 181-201. Ed. Kluwer Academic Publishers, USA, 2002.

KOLB, D. A. *et al. Psicologia Organizacional: Uma Abordagem Vivencial.* São Paulo, Atlas, 1986.

KLEENE, S. C. Recursive functions and intuitionistic mathematics. In: *Proceedings of the International Congress of Mathematicians.* Cambridge, Massachusetts, USA, Vol. 2, p. 679–685, 1952.

KLEENE, S. C. *The Work of Kurt Gödel.* The Journal of Symbolic Logic, Vol. 41, p. 761-778. 1976.

KNUTH, D. *The Art of Computer Programming.* V.1, Addison-Wesley, 3rd ed, 1968.

LEVY, P. *A inteligência Coletiva: por uma antropologia do ciberespaço.* Ed. Loyola, 1994.

LINDER et al., [Linder, S. P., Nestrick, B. E., Mulders, S., Lavalle, C. L.] *Facilitating Active learning with inexpensive Móbile Robots.* The journal of Computing in small colleges. Anais do 6º CCSC. Northeastern conference on the journal of computing in small coleges, Vol. 16, 2001.

LISTER R. On Blooming First Year Programming, and its Blooming Assessment. Proceedings of the Australasian Conference on Computing Education, ACM International Conference Proceeding Series, Vol. 8. Melbourne, Australia. p 158-162, 2000.

MALISANI, E. *Los Obstáculos Epistemológicos em el Desarrollo del Pensamento Algébrico*. Revista IRICE, nº 13, Argentina, 1999.

MANSO, A. *PORTUGOL, Simulador de Linguagem Algorítmica*. Versão 2.2, 2006. Disponível em http://orion.ipt.pt/~manso/Portugol/ . Acessado pela última vez em 15/10/2008.

MAURER, S., *"What is Discrete Mathematics? The Many Answers"*. Discrete Mathematics in the School- DIMACS, V. 36. p. 121-132, 1997.

McCRACKEN, M.; WATERS R. *Why? When na Otherwise Successful Intervention Fails*. ACM SIGCSE (Technical Symposium on Computer Science Education), Vol. 31. 1999.

OLIVEIRA, C. O.; COSTA, J. W.; MOREIRA, M. *Ambientes Informatizados de Aprendizagem*. Papirus, 2001.

MEC. Comissão de Especialistas de Ensino de Computação e Informática - MEC/SESu. Disponível em http://www.inf.ufrgs.br/mec/ . Acessado pela última vez em 02/02/2009.

PAIS, L. C. *Didática da Matemática: uma análise da influência francesa.* Belo Horizonte: Autêntica, 2002.

PORTO, C.; RÉGNIER, K. *O ensino superior no mundo e no Brasil – condicionantes, tendências e cenários para o horizonte 2003-2005: Uma abordagem exploratória*. Dezembro, 2003. Disponível em: <www.mec.gov.br/sesu>. Acessado pela última vez em 07/08/2008.

POST, E. *Formal Reductions of the General Combinatorial Decision Problem.* American Journal of Mathematics, Vol. 65, p.197–215, 1943.

POWERS, K. D. & POWERS D. T. *Making Sense of Teaching Methods in Computing Education*. 29th ASEE/IEEE Frontiers in Educational Conference. p. 11B3/30-11B3/35, Vol.

1, 1999.

RESNICK, M.; MARTIN, F., SARGENT, R.; SILVERMAN, B. *Programmable Bricks: Toys to Think With*. IBM Systems Journal, Vol. 35, N. 3&4, 1996.

SAGASTUME, M.; BAUM, G. *Problemas, lenguajes y algoritmos*. Campinas: Unicamp, Centro de Lógica, Epistemologia e História da Ciência, 2. ed., 2003.

SALVETTI, D. D; BARBOSA, L.M. *Algoritmos*. São Paulo: Makron Books, 1998.

SCHUBRING , G. *A Noção de Multiplicação: Um "Obstáculo" desconhecido na História da Matemática*. Revista Bolema, Ano 15, nº 18, p. 26 a 50, 2002.

SCHOENFELD, A. H. *Learning to Think Mathematically: Problem Solving, Metacognition, and Sense Making in Mathematics, Handbook for Research on Mathematics Teaching and Learning* (D. Grouws, Ed.), New York: MacMillan, Chapter 15, p. 334-370, 1992.

SCHOENFELD, A H. Porquê toda esta agitação acerca da resolução de problemas? In: *Investigar para aprender matemática.* ABRANTES, P.; LEAL, L. C.; PONTE, J. P. (Orgs.). p. 61 – 71. Portugal, APM, 1998.

SETTI, M. G.; CIFUENTES, J. C. *Os Obstáculos Epistemológicos no Pensamento Computacional*. 8º Simposio de Educación Matemática. Buenos Aires, 2006.

SETTI, M. G.; CIFUENTES, J. C. *O Pensamento Matemático Elementar como um Obstáculo Epistemológico para o Pensamento Computacional.* Anais do 2º Simpósio Nacional de Tecnologia e Sociedade, 2007, Curitiba. UTFPR, v. I. p. 1-7, 2007.

SHEPHERDSON, J. C.; STURGIS, H. E. *Computability of Recursive Functions*. Journal of the ACM, n.10, p. 217-255. 1963.

SIERPINSKA, A. *Obstacles Epistemologiques Relatifs a la Notion de Limite*. Recherches en Didactique des Mathematiques, Vol. 6, n.1, p. 5-67, 1985.

SOBRINHO, J. Z., *Aspectos da Tese de Church-Turing*. Revista Matemática Universitária, n. 6, p. 1-23. USP – São Paulo, 1987.

SOUZA, C. M.*VisuAlg - Editor e Interpretador de Pseudocódigos Objetivos.* Disponível em: http://www.apoioinformatica.inf.br/visualg/objetivos.htm. Acessado pela última vez em 14/10/2008.

SPAGNOLO, F. *Obstacles Epistemologiques: et Postulat d'Eudoxe – Archimede.* Tesis de Doctorado. Universidad de Bordeaux I. Quademi di Ricerca Didattica G.R.I.M., Supplemento n. 5. Publicada por el Atelier de Reproduction dês Théses Microfiches (BP – 38040 Grenoble.

Cedex 9 – Francia), 1996.

TURING, A. M. *On Computable Numbers, with an application to the Entscheidungsproblem.* Proc. London, Math. Soc., Vol. 42. p. 230-265, 1936.

WANG, H. *From Mathematics to Philosoph.*, Routledge & Kegan Paul, London,1974.

WINSLOW, L. E. *Programming Pedagogy – A Psychological Overview.* Anais do 27º ACM SIGCSE (Technical Symposium on Computer Science Education), USA, 1996.

WIRTH, N. *Algoritmos e Estruturas de Dados*. Rio de Janeiro: Prentice Hall do Brasil, 1989.

WOLZ, U. *Teaching Design and Project Management with Lego RCX Robots.* Anais do 32º ACM SIGCSE (Technical Symposium on Computer Science Education), USA, 2001.

YIN, R. K. *Estudo de Caso: Planejamento e Métodos*. 3. ed. Bookman, 2005.

ZIEGLER, U.; CREWS, T. *An Integrated Program Development Tool for Teaching and learning How to Program.* Anais do 30º ACM SIGCSE (Technical Symposium on Computer Science Education), USA, 1999..

Bibliografia Complementar

ALG-PROG. Lista de Discussão sobre Algoritmo e Programação. Disponível em https://grupos.ufrgs.br/mailman/listinfo/alg-prog-l. Acessado pela última vez em 30/09/2008.

ALMOULOUD, S. A. Registros de Representação Semiótica e Compreensão de Conceitos Matemáticos. In: Alcântara Machado, Silvia D. (Ed.) *Aprendizagem matemática:* Representação Semiótica. São Paulo: Papirus, 125-147, 2003.

ARAÚJO, C. *Um Salto Quântico No Infinito.* Jornal Estado de Minas de 4 de abril de 1998, na seção Pensar.

BORBA,M.C. Tecnologias Informáticas na Educação Matemática e Reorganização do pensamento. In:Bicudo, M.A.V. (Org.), *Pesquisa em Educação Matemática:* Concepções & Perspectivas. Ed. Unesp, 1999.

CLANCY, M. J.; LINN, M. C. *Patterns and Pedagogy*. Anais do 30º ACM SIGCSE (Technical Symposium on Computer Science Education), V. 31, 1999.

DAMM, R. F. Registros de Representação. In: *Educação Matemática:* Uma Introdução, p. 135-154. São Paulo: Educ, 1999.

GPIMEM – Grupo de Pesquisa em Informática. *Outras Mídias e Educação Matemática*. Disponível em http://www.rc.unesp.br/igce/pgem/gpimem.html. Acessado pela última vez em 10/11/2008.

HUNDHAUSEN, C. D.; DOUGLAS, S. A.; STASKO, J.A. *A Meta-Study of Algorithm Visualization Effectiveness*. Journal of Visual Language and Computing. 2002. Disponível em: http://eecs.wsu.edu/%7Eveupl/pub/JVLC-AVMetaStudy.pdf. Acessado pela última vez em 10/03/2008.

IFUSP. *Ciência à Mão, Portal de Ensino de Ciências.* Experimentoteca Ludoteca • Instituto de Física da USP. Disponível em http://www.ludoteca.if.usp.br/tudo/tex.php?cod=_fibonacciproblemadoscoelhos. Acessado pela última vez em 02/02/2008.

IGLIORI, S.B. *Educação matemática: A Noção de Obstáculo Epistemológico e a educação matemática.* São Paulo: EDUC, 2002.

METZGER, R. *Debugging by Thinking*: A Multidisciplinary Approach. Digital Press, 2003.

NEGRELLI, G. L. *Uma reconstrução epistemológica do processo de modelagem matemática para a educação (em) matemática.* Tese de doutorado apresentado ao setor de Educação da Universidade Federal do Paraná, UFPR, Brasil, 2008.

NEWELL, A.; SIMON, H. A. *Human Problem Solving*. Englewood Cliffs, NJ: Prentice Hall, 1972.

PLACCO, V. M. *Psicologia e Educação. Revendo Contribuições.* São Paulo: EDUC, 2000.

REZENDE, P. A. D.; *A Crise nos Fundamentos da Matemática e a Teoria da Computação.* Palestra proferida no Seminário de Filosofia da Unb, 1999. Disponível em: http://www.cic.unb.br/docentes/pedro/trabs/a_crise.pdf. Acessado pela última vez em 15/10/2007.

SBC (Sociedade Brasileira de Computação). *II Curso de Qualidade de Cursos de Graduação da Área de Computação e Informática.* Curitiba-PUCPR, 2000.

SILVA, L. M.; BAROLLI, E. *Registros de Representação Semiótica na Resolução de Problemas*. Anais do Seminário Internacional de Pesquisa em Educação Matemática, Águas de Lindóia – SP, 2006.

STRAUHS, F.R., *Gestão do Conhecimento em Laboratórios de Ensino, Pesquisa e Desenvolvimento.* Tese de Doutorado em Engenharia de Produção. UFSC, Brasil, 2003.

TRINDADE, J. A. O. Os *obstáculos Epistemológicos e a Educação Matemática.* Dissertação de mestrado apresentada ao Depto. de Educação da UFSC, 1996.

TIKHOMIROV, O. K. The Psychological consequences of computerization. In: WERTTSCH, J.V. (Ed.) *The concept of activity in soviet psychology*. New York: M.E. Sharpe. Inc, 1981.

Apêndice I

I.1- Avaliação Aplicada no Semestre 01/2007

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria do Campus Curitiba
Gerência de Ensino e Pesquisa
Departamento Acadêmico de Informática
Disciplina: Lógica de Programação
Profª. Mariângela Setti 30/03/07.

Aluno(a): _____ Código: _____

1. Considere as seguintes proposições:

   p: Os agricultores se mobilizam.

   q: Os políticos nada fizeram.

   r: A reforma agrária continua sem solução.

   S: Os sem-terra apelam para o presidente da República.

   Agora, simbolize as seguintes sentenças:

   a) Os agricultores se mobilizam e os políticos nada fizeram ou a reforma agrária continua sem solução.
   b) Se agricultores não se mobilizam, então a reforma agrária continua sem solução.
   c) Os agricultores se mobilizam e os políticos nada fizeram ou não é o caso de a reforma agrária continuar sem solução e os sem-terra apelam para o Presidente da República.
   d) Se os agricultores se mobilizam, então, os políticos nada fizeram e a reforma agrária continua sem solução.
   e) Se os políticos nada fizeram, então, se a reforma agrária continua sem solução, então, os sem-terra apelam para o Presidente da República.

2. Sabendo que as proposições “p” e “q” são verdadeiras e que as proposições “r” e “s” são falsas, determinar o valor lógico (V ou F) das seguintes proposições:

   a) $(p \wedge q) \rightarrow (r \vee s)$
   b) $(q \vee s) \wedge (p \vee r)$
   c) $\sim s \rightarrow (p \wedge q)$

d) $(r \rightarrow s) \wedge (p \vee q)$

e) $\sim[(p \rightarrow r) \vee (q \rightarrow s)]$

3. Sejam as proposições:

p: São Paulo está na América do Sul.

q: Começou o inverno.

Determine o valor lógico de:

a) $(\sim p \wedge q) \vee \sim q$

b) $p \rightarrow (q \rightarrow \sim p)$

c) $(p \leftrightarrow q) \vee p$

d) $(q \vee p) \rightarrow q$

4. Verificar, utilizando tabela verdade, se as proposições seguintes são tautológicas, contraditórias ou contingenciais:

a) $(p \wedge r) \rightarrow (\sim q \vee r)$

b) $(p \vee q) \leftrightarrow p \wedge q$

c) $(p \leftrightarrow q) \rightarrow (\sim q \vee p)$

d) $(p \rightarrow q) \vee (q \vee \sim r)$

5. Demonstre, utilizando tabelas-verdade, as seguintes relações de equivalência:

a) $p \leftrightarrow (p \wedge q) \Leftrightarrow p \rightarrow q$

b) $q \rightarrow p \Leftrightarrow \sim p \rightarrow \sim q$

c) $p \vee (p \wedge q) \Leftrightarrow p$

d) $q \leftrightarrow (p \vee q) \Leftrightarrow p \rightarrow q$

6. Faça um algoritmo para calcular a média dos alunos de sua turma (máximo 30) no semestre 01/2007. O algoritmo deve:

- ler as notas das 7 disciplinas do primeiro período de cada aluno, fazendo a consistência para que a nota seja ≥ 0.
- Calcular a média de cada aluno, e escrever na tela;

- Encontrar a maior média da turma;
- Verificar o percentual de alunos que tiveram alguma reprovação e ao final escreva o valor na tela.

7. No correio local há somente selos de 3 e 5 centavos. A taxa mínima para correspondência é de 8 centavos. Deseja-se determinar o menor número de selos de 3 e de 5 centavos que completem o valor de uma taxa fornecida pelo usuário, que deve obrigatoriamente ≥ 8 (fazer a consistência). Faça um algoritmo e o teste de mesa para resolver o problema, utilize as mensagens adequadas.

I.2- Avaliação Aplicada no Semestre 02/2007

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria do Campus Curitiba
Gerência de Ensino e Pesquisa
Departamento Acadêmico de Informática
Disciplina: Lógica de Programação – Profª. Mariângela Setti 20/09/07.

Aluno(a): _____ Código: _____

OBS: Nota da prova = 8,0 + 2,0 referente às listas de exercícios e ao trabalho.

1. Encontre o antecedente e o consequente de cada uma das proposições a seguir.
    a. O crescimento sadio das plantas é consequência da quantidade suficiente de água.
    b. O aumento da disponibilidade de informação é uma condição necessária para um maior desenvolvimento tecnológico.
    c. Serão introduzidos erros apenas se forem feitas modificações no programa.
    d. A economia de energia para aquecimento implica boa insulação ou vedação de todas as janelas.

2. Sejam **p**, **q** e **r** as seguintes proposições:

    p: Rosas são vermelhas.

    q: Violetas são azuis.

    r: Açúcar é doce.

    Escreva as proposições compostas a seguir em notação simbólica:

    a. Rosas são vermelhas e violetas são azuis.
    b. Rosas são vermelhas, e ou bem violetas são azuis ou bem açúcar é doce.
    c. Se violetas são azuis, rosas são vermelhas e açúcar é doce.
    d. Rosas são vermelhas se somente se violetas não forem azuis e se açúcar for amargo
    e. Rosas são vermelhas e, se açúcar for amargo, então ou violetas não são azuis ou açúcar é doce.

3. Use p, q e r conforme definido no exercício 2 (acima) para escrever as seguintes

proposições compostas em português:

a. $q \vee \sim r$
b. $\sim q \vee (p \rightarrow r)$
c. $(r \wedge \sim p) \leftrightarrow q$
d. $\sim(q \wedge \sim r) \rightarrow p$
e. $r \leftrightarrow (p \vee q)$

4. Diga qual a proposição que define a tabela-verdade a seguir:

P	Q	?
V	V	F
V	F	V
F	V	V
F	F	F

a. $p \rightarrow q$
b. $q \rightarrow p$
c. $p \leftrightarrow q$
d. $\sim(p \leftrightarrow q)$
e. $\sim p \rightarrow q$

5. Verificar, utilizando tabela verdade, se as proposições seguintes são tautológicas, contraditórias ou contingenciais:

e) $[(p \wedge r) \wedge (\sim q \vee r)] \rightarrow (q \wedge r)$
f) $(p \leftrightarrow q) \rightarrow (\sim q \vee p)$
g) $[p \wedge (q \vee \sim p)] \leftrightarrow p$
h) $[r \vee (p \wedge q)] \rightarrow r$

6. Faça um algoritmo que leia o ano de nascimento de uma pessoa, calcule e mostre sua idade, considerando a ano atual. Verifique e mostre se ela já tem idade para votar (idade > 16), e para tirar carteira de habilitação (idade > 18). O algoritmo deve emitir as mensagens adequadas, informando as possibilidades do usuário. Faça o algoritmo primeiro

identificando as entradas, saídas e o processamento (E/ S/ P) e depois em portugol.

7. O IMC – Índice de Massa Corporal é um critério da Organização Mundial de Saúde para dar uma indicação sobre a condição de peso de uma pessoa adulta. A fórmula é IMC = peso / (altura)^2. Elabore um fluxograma que tenha como entrada o peso e a altura de um adulto e mostre sua condição, segundo a tabela abaixo:

IMC em adultos	Condição
Abaixo de 18,5	Abaixo do peso
Entre 18.5 e 25	Peso normal
Entre 25 e 30	Acima do peso
Acima de 30	Obeso

I.3- Avaliação Referente à Elaboração do Raciocínio Matemático x Computacional

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Diretoria do Campus Curitiba
Gerência de Ensino e Pesquisa
Departamento Acadêmico de Informática
Disciplina: Lógica de Programação – Profª. Mariângela Setti 11/04/08.

Aluno(a): _____ Código: _____

Sejam as proposições:

p: O rato entrou no buraco

q: O gato seguiu o rato

Forme sentenças, em linguagem natural, que correspondam às proposições seguintes: (0,5)

a) $\sim (p \wedge q)$

b) $\sim p \wedge \sim q$

c) $p \leftrightarrow \sim q$

d) $q \rightarrow (p \wedge q)$

Determine o valor lógico de p, isto é v(p), em cada um dos seguintes quesitos, sabendo que: (0,5)

a) $v(q) = V$ e $v(p \leftrightarrow q) = F$

b) $v(q) = V$ e $v(p \wedge q) = V$

c) $v(q) = F$ e $v(p \rightarrow q) = F$

d) $v(q) = V$ e $v(p \vee q) = V$

3. Sabendo que o valor lógico de “p” e o valor lógico de “r” são ambos verdadeiros, isto é, $v(p) = v(r) = V$ e que o valor lógico de “q” e o valor lógico de “s” são ambos falsos, ou seja, $v(q) = v(s) = F$, julgar, quanto ao valor lógico, cada uma das seguintes proposições: (1,0)

a) $(p \vee q) \leftrightarrow (r \vee \sim s)$

b) $(r \rightarrow s) \rightarrow (\sim p \rightarrow q)$

c) $(p \rightarrow \sim q) \leftrightarrow [(p \wedge r) \vee s]$

d) $(p \wedge \sim s) \wedge (\sim s \vee r)$

Verificar, utilizando tabela verdade: (1,0)

a. se a proposição $[p \rightarrow (q \wedge r)] \rightarrow (p \rightarrow r)$ é uma tautologia;

b. se a proposição $[(p \wedge q) \vee (p \rightarrow q)] \leftrightarrow (\sim p \wedge \sim q)$ é contigencial;

c. se a proposição $\sim(p \wedge q) \rightarrow (\sim p \vee \sim q)$ é uma contradição.

5. Demonstre, utilizando tabelas-verdade, as seguintes relações de equivalência: (1,0)

a. $p \vee q \Leftrightarrow (p \rightarrow q) \rightarrow p$

b. $p \vee (q \wedge r) \Leftrightarrow (p \vee q) \wedge (p \vee r)$

c. $(p \leftrightarrow q) \vee \sim p \Leftrightarrow p \rightarrow q$

d. $(p \rightarrow q) \rightarrow r \Leftrightarrow (p \wedge \sim r) \rightarrow \sim q$

6. Utilizando tabelas-verdade, verifique se existem as relações de implicação lógica seguintes (1,0):

a) $p \leftrightarrow q \Rightarrow q \rightarrow p$

b) $\sim p \vee (\sim q \rightarrow p) \Rightarrow \sim(p \vee \sim q)$

c) $(p \rightarrow q) \vee (r \rightarrow \sim q) \Rightarrow (r \rightarrow p)$

d) $(p \vee q) \wedge (p \vee \sim q) \Rightarrow p$

7. Uma academia de ginástica armazena algumas informações sobre os seus clientes (nome, sexo, altura), considere que a academia tem no máximo 1000 clientes. Faça um algoritmo, utilizando portugol e fluxograma que leia essas informações e determine o peso ideal do cliente, sabendo que a fórmula para o cálculo do peso ideal é dada a seguir. Faça a consistência para que o campo sexo seja = F ou M. (2,0)

- Masculino: (72.7 * altura) -58
- Feminino: (62.1 * altura) -44.7

8. Um fazendeiro tem um par de coelhos recém-nascidos em um ambiente fechado. Considerando que, de um modo natural, a cada mês ocorre a produção de um par e um par começa a produzir coelhos quando completa dois meses de vida. Desejamos saber quantos pares de coelhos o fazendeiro irá possuir ao final de N meses”[8]. Pede-se:

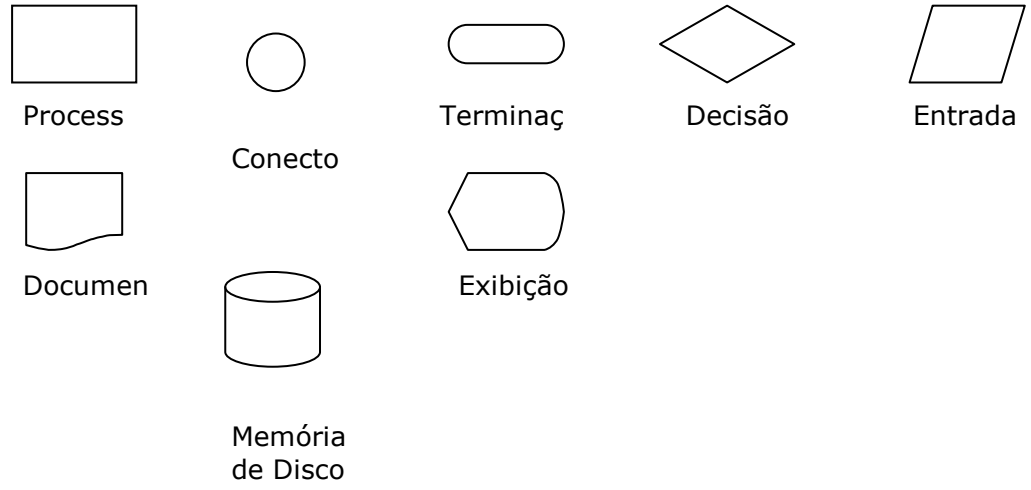
---

[8] Leonardo de Pisa, filho de Guglielmo Bonacci, daí ser chamado de Fibonacci (*filius Bonacci*), apesar de ter sido um dos introdutores dos números Hindu-arábicos na Europa e principalmente, o introdutor do zero, ficou mais conhecido pela sequência que leva seu nome. Escreveu um livro denominado *Liber Abacci*, no qual propôs um problema que está nas páginas 123 e 124, denominado Problema dos pares de coelhos (*paria coniculorum*), que acaba gerando sua famosa sequência (IFUSP, 2009). Este problema é um problema clássico no ensino de introdução à programação.

- O esboço gráfico do problema.
- A solução matemática, chegando à fórmula que define a produção de coelhos a partir do terceiro mês.
- Baseado nas suas descobertas nos itens anteriores, faça um algoritmo em Portugol para encontrar o N-ésimo termo da sequência, sendo que N deve ser digitado pelo usuário.
- Faça o teste de mesa para N = 6.

Anexo

Símbolos utilizados em um fluxograma:



Apêndice II

II.1- Plano de Ensino da Disciplina de Lógica de Programação

UTFPR UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ	Ministério de Educação Universidade Tecnológica Federal do Paraná Campus de Curitiba Gerência de Ensino e Pesquisa Departamento Acadêmico de Informática	CÓDIGO DA UNIDADE CURRICULAR **IF51A-M71**
PLANO DE ENSINO: Lógica de Programação		

01	CURSO: **Tecnologia em Sistemas para Internet**	ÁREA: INFORMÁTICA
	MÓDULO B: Desenvolvimento de Sistemas em Ambiente Distribuído	PERÍODO: 01/2008
N.º de horas-aula semanal: 5 N.º de horas-aula semestral: 90	N.º total de aulas teóricas: 20 N.º total de aulas práticas: 34 N.º total de aulas de laboratório: 36	

02	OBJETIVO GERAL DO CURSO
O Curso Superior em Tecnologia em Desenvolvimento de Sistemas Distribuídos tem como finalidade profissional egresso do curso atenderá os seguintes postos de trabalho: • Desenvolvedor de sistemas distribuídos • Instalação e administração rede de computadores e seus serviços • Desenvolvimento e Gerencia de projetos e aplicações Distribuídas • Desenvolvimento e Gerencia de projetos e aplicações de sistemas móveis e sem fio • Coordenador de projetos de software Empreendedor do próprio negócio	

03	PERFIL PROFISSIONAL DOS EGRESSOS
	O egresso do Curso Superior de Tecnologia em Desenvolvimento de Sistemas Distribuídos deverá ser capaz de: • Desenvolver aplicativos em linguagens procedurais e orientado à objeto; • Projetar sistemas de software utilizando ferramentas de apoio; • Desenvolver aplicações dinâmicas para ambiente Web; • Desenvolver aplicações para dispositivos móveis e sem fio; • Desenvolver aplicações baseadas em objetos distribuídos; • Instalar; configurar e administrar sistemas operacionais; • Instalar; configurar e administrar equipamentos de redes; • Instalar; configurar e administrar serviços de rede; • Integrar sistemas corporativos com aplicativos baseados em dispositivos móveis e sem fio; • Integrar sistemas legados com sistemas atuais. • Integrar sistemas heterogêneos;

- Iniciar e gerenciar um empreendimento;
- Coordenar e gerenciar projetos de software e sistemas de informação;
- Analisar o desempenho e demais características de rede de computadores para otimizar o seu funcionamento (*tunning*);
- Desenvolver serviços Web e de suporte para comércio eletrônico;
- Utilizar recursos de segurança para a proteção e monitoramento de recursos de rede;
- Monitoração e gestão de segurança de ambientes distribuídos;

04	COMPETÊNCIAS A SEREM CONSTRUÍDAS NO MÓDULO Formam a competência de **Desenvolvimento de Sistemas em Ambiente Distribuído** as seguintes unidades curriculares • LÓGICA DE PROGRAMAÇÃO • CONSTRUÇÃO DE PÁGINAS WEB • COMUNICAÇÃO LINGÜÍSTICA • FUNDAMENTOS MATEMÁTICOS DA COMPUTAÇÃO • PROJETO DE SISTEMAS DE BANCO DE DADOS • ESTRUTURA DE DADOS, PESQUISA E ORDENAÇÃO • ANÁLISE DE SISTEMAS • LINGUAGEM DE PROGRAMAÇÃO • LINGUAGEM DE PROGRAMAÇÃO VISUAL • TECNOLOGIA DE ORIENTAÇÃO A OBJETOS E JAVA • METODOLOGIA DA PESQUISA CIENTÍFICA E TECNOLÓGICA • INTERFACE HOMEM MÁQUINA • DESENVOLVIMENTO DE APLICAÇÕES WEB • TECNOLOGIA DE BANCO DE DADOS DISTRIBUÍDOS • PROGRAMAÇÃO PARA DISPOSITIVOS MÓVEIS E SEM FIO • SERVIÇOS DE SUPORTE A APLICAÇÕES DISTRIBUÍDAS • SISTEMAS OPERACIONAIS DISTRIBUÍDOS • WEB SERVICES XML • ENGENHARIA DE SOFTWARE Formam a competência **Núcleo Comum** as seguintes unidades curriculares: • SISTEMAS DISTRIBUÍDOS • ESTATÍSTICA • GERÊNCIA E CONFIGURAÇÃO DE SERVIÇOS INTERNET • TECNOLOGIAS MÓVEIS E SEM FIO

05	FUNDAMENTAÇÃO LEGAL:
	Lei Federal n.º 9.394/96. Decreto Federal n.º 2.208/97. Parecer n.º 21/99 COENS de 21/01/99. Parecer n.o 3/99 CODIR de 19/03/99. Reconhecimento em 19/12/03.

06	UNIDADE CURRICULAR: *Lógica de Programação*

07	HABILIDADES:
1.	Relacionar os fundamentos da lógica formal com a construção de algoritmos e a ciência da computação.
2.	Desenvolver raciocínio computacional, para elaborar algoritmos, através de uma linguagem escrita (ex. Portugol) e uma gráfica (ex. Fluxograma) seguindo os preceitos da programação estruturada.

08	**Competência**	**Bases Tecnológicas(Ementa)**	**NA**	**ME**	**RD**	**FA**
Utilizar lógica matemática para expressar raciocínios e construir algoritmos de maneira formal.		Cálculo proposicional. Conectivos lógicos fundamentais. Equivalência e Implicação lógica. Álgebra proposicional. Cálculo dos predicados. Procedimentos decisão e validade.	15	1, 2	1, 4, 8	2, 8
		Construção de algoritmos. Estruturas básicas de decisão e controle. Operadores aritméticos, relacionais e lógicos. Conceitos de programação estruturada e modular. Procedimentos e funções. Atividades em laboratório.	65	1, 3, 4, 5	1, 8	2, 8

09	PROJETO INTEGRADOR

10	BIBLIOGRAFIA
	1. ASCENCIO, A.F.;CAMPOS, E.A. *Fundamentos da programação de Computadores*. São Paulo: Prentice Hall, 2002. 3. FARRER, H.; Algoritmos Estruturados. São Paulo: LTC, 1999. 4. FORBELLONE, A. L.; Eberspacher, H.F. *Lógica de Programação*. São Paulo: Makron Books, 2000. 5. GERSTING, J. L. *Fundamentos Matemáticos da Computação.* São Paulo: LTC, 1995. 6. GUIMARÃES, A. M. LAGES, N. A. C. *Algoritmos e Estruturas de Dados.* São Paulo: LTC. 7. SUCHEUSKI, M. *Desenvolvedor Profissional – Algoritmos.* Curitiba: Lísias, 1996.

II.2- Plano de Aula da Disciplina de Lógica de Programação

UTFPR UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ	Ministério de Educação Centro Federal de Educação Tecnológica do Paraná Unidade de Curitiba Gerência de Ensino e Pesquisa Departamento Acadêmico de Informática	***CÓDIGO DA UNIDADE CURRICULAR*** ***IF51C- M71***
	PLANO DE AULA **CST EM Desenvolvimento de Sistemas Distribuídos**	

01	**UNIDADE CURRICULAR:** *Lógica de Programação*

Semestre/ano	Aulas Teóricas	Aulas de Laboratórios	Aulas Práticas	Carga horária prevista
	20	36	34	90

Bases Tecnológicas / Conteúdo	NA	ME	RD	FA
Conceito intuitivo de algoritmos	5	1, 2	1, 4	
Introdução à Lógica Matemática. Histórico. Fórmulas proposicionais. Conceituar proposição e valor-verdade. Princípios fundamentais da lógica. Conceituar sentenças compostas, conectivos lógicos, tabela-verdade.	3	1	1	
Com tabelas-verdade, detalhar os conectivos lógicos: negação, conjunção, disjunção inclusiva, disjunção exclusiva, condicional e bicondicional. Tautologia, Contradição.	3	1	1, 8	
Funções Lógicas com Excel	4	2	4	
Introdução a algoritmos. Variáveis e constantes. Tipos de Dados. Expressões Aritméticas, Operadores Relacionais e Lógicas. Programação estruturada,e os conceitos de programação *top-down*.	3	1, 5	1, 8	2, 8
Apresentar a linguagem Portugol e Fluxograma. Comandos de entrada e saída de dados, comandos de atribuição, estruturas condicionais.	5	1, 3, 4	1, 8	2, 8
Apresentar em Portugol e Fluxograma os comandos de repetição.	10	1, 3, 4	1, 8	2, 8
1ª Avaliação Parcial				2, 8
Comandos de repetição.	10	1, 3, 4	1, 8	2, 8
Algoritmos e estruturas de dados homogênias (vetores e matrizes).	20	1, 3, 4	1, 8	2, 8
2ª Avaliação Parcial				2,8
Algoritmos e estruturas de dados heterogêneas (registros).	12	1, 3, 4	1, 8	2, 8

Conceituar funções e procedimentos. Escopo de variáveis. Conceituar funções recursivas.	15	1, 3, 4	1, 8	2, 8
3ª Avaliação Parcial				2,8
4ª Avaliação Substitutiva				2, 8

03	**ORIENTAÇÕES GERAIS**

a) Freqüência Mínima = 75% de NA dadas
b) Nota Final = (Média ∑ N.º Av do Semestre)
c) Identificar no Plano as datas das Avaliações ao longo do semestre a serem realizadas

04	**APRENDIZAGEM E AVALIAÇÃO**

ME: Metodologia de Ensino	***RD: Recursos Didáticos***	***FA: Formas de Avaliação***
1 - Expositiva - dialogada 2 - Atividade de laboratório 3 - Trabalho individual 4 - Trabalho em grupo 5 - Pesquisa 6 - Dramatização 7 - Projeto 8 - Debate 9 - Estudo de caso 10 - Seminário 11 - Painel integrado 12 - Visita técnica 13 - Brainstorming 14 - Outros	1 - Transparências 2 - Slides 3 - Videocassete 4 - Computador 5 - Mapas / catálogos 6 - Laboratório 7 - Impressos (apostila) 8 - Quadro de giz 9 - Outros	1 - Prova objetiva 2 - Prova dissertativa 3 - Prova prática 4 - Palestra 5 - Projeto 6 - Relatório 7 - Seminário 8 - Outros Obs.: Nota final= (Média ∑ N.º Av) Av= Avaliação