Sumário

Prefácio

A quem se destina este livro?

Este livro destina-se a um curso introdutório de lógica de programação, especialmente para aqueles ministrados em escolas de Engenharia. Um dos principais problemas encontrados pelo estudante de Engenharia em um primeiro curso de lógica de programação é a carência de textos que abordem de forma direta e clara as etapas necessárias para suportar o processo de resolução de problemas (computacionais ou não), a saber: a *análise*, com a identificação e solução de subproblemas, e a *síntese*, união das soluções encontradas para compor a solução do problema original. O resultado dessas etapas é sintetizado em passos que devem ser seguidos em determinada ordem e que constituem os *algoritmos*.

Abordagem empregada

Pretende-se aqui seguir uma apresentação incremental dos tópicos. Inicialmente são propostos problemas simples que envolvem raciocínio lógico e que possuem solução livre, de modo a ambientar e a incentivar o estudante na descrição dos passos elementares necessários à resolução de problemas. Isso é fundamental, pois grande parte dos estudantes que tem um primeiro contato com lógica de programação apresenta deficiências na organização de suas soluções e em abstrações. Além disso, neste primeiro contato, um processo genérico de solução de problemas é apresentado de maneira a fornecer um conjunto de dicas ou heurísticas que podem ser aplicadas em todos os problemas a serem resolvidos, fortalecendo assim o processo de abstração, essencial em programação.

A seguir são apresentados os conceitos de computação e computadores. Embora um primeiro curso de lógica de programação possa ser ministrado sem referências a como um computador é organizado e como funciona, verifica-se na prática que esse enfoque não é adequado. Como se sabe, o grande problema do estudante nesses cursos

introdutórios é a abstração de procedimentos e dados. Nesse ponto apresenta-se uma arquitetura de computador bem simples, baseada na arquitetura de Von Neumann, para fixar de modo tangível os conceitos relacionados a instruções e dados operados em computadores. Objetiva-se aqui que o estudante futuramente consiga relacionar os aspectos abstratos de computação, tais como variáveis, estruturas de programas e decomposição funcional com sua implementação. Essa parte serve ainda como incentivo para a necessidade de se descrever algoritmos antes de sua implementação propriamente dita.

Depois, e acompanhando todo o livro, emprega-se uma notação formal para a solução de problemas. Utiliza-se neste texto a descrição de algoritmos sob a forma de *fluxogramas* baseados na norma ISO 5807/1985. Os fluxogramas são compostos por símbolos básicos que representam as menores partes em um processo de solução: estruturas seqüenciais, de decisão e de repetição. O uso de fluxogramas nesta obra é justificado pelo fato de que o engenheiro tem a obrigação de desenvolver um raciocínio lógico bem-estruturado e que o fluxograma ainda representa uma poderosa ferramenta para a verificação e teste da lógica empregada na solução de problemas. A utilização de fluxogramas em Engenharia é ampla: de descrições de programas até descrições de processos de fabricação ou processos químicos, seu emprego é similar e regido única e exclusivamente pela lógica utilizada na composição de seus blocos, até se alcançar a solução de um determinado problema.

Além do uso de fluxogramas, são apresentadas ainda duas outras formas conhecidas para a representação de algoritmos: diagramas de *Nassi-Schneidermann* e o pseudocódigo baseado na língua portuguesa, o *Portugol*. Os diagramas de Nassi-Schneidermann empregam uma representação em "caixas" aninhadas, em que cada uma é relacionada a um determinado tipo de comando ou estrutura de programação. Já o Portugol usa uma descrição textual e estruturada da solução de um problema na qual os comandos são descritos por palavras-chave reservadas e extraídas da língua portuguesa.

Descrição dos capítulos

No Capítulo 1 são apresentados os conceitos básicos sobre modelagem de problemas em Engenharia e como organizar suas soluções utilizando passos elementares. Faz-se aqui um prelúdio ao estudo dos algoritmos, com uma descrição de métodos para auxiliar o estudante no processo de identificação e resolução de problemas, bem como a proposição de problemas de lógica com solução livre para ambientar o estudante nesse processo.

No Capítulo 2 são discutidos os conceitos de computação e computadores. Inicia-se com a discussão da origem da palavra computação, seu significado e aplicações. A seguir são discutidos os conceitos básicos sobre a organização de computadores utilizando a

arquitetura de Von Neumann. Um computador hipotético com instruções simplificadas é apresentado de forma a proporcionar ao estudante simulações de como as instruções e os dados são realmente processados.

Os conceitos de algoritmo e fluxograma são formalizados no Capítulo 3. São discutidos o conceito e as propriedades de um algoritmo, a representação de algoritmos por fluxogramas, como criar um fluxograma utilizando os símbolos básicos da norma ISO 5807/1985, bem como convenções para os tipos de dados, os nomes de variáveis e operadores.

Já no Capítulo 4 formalizam-se as estruturas de programação. São apresentados os nomes e as topologias das estruturas típicas de um programa: as estruturas seqüenciais, de decisão e de repetição. Apresentam-se ainda nesse capítulo duas outras formas de representação de algoritmos: os diagramas de Nassi-Schneidermann e a pseudolinguagem Portugol.

É apresentado, no Capítulo 5, o conceito de variáveis indexadas e seu uso. As variáveis indexadas são aquelas que referenciam de forma ordenada uma seqüência de dados homogêneos. Separam-se aqui, para melhor compreensão, o conceito e a utilização de variável indexada unidimensional (ou vetor) do conceito de variável indexada bidimensional e multidimensional. Com essa separação, espera-se que o estudante consiga estender os conceitos e operações relacionados a variáveis indexadas unidimensionais para dimensões maiores.

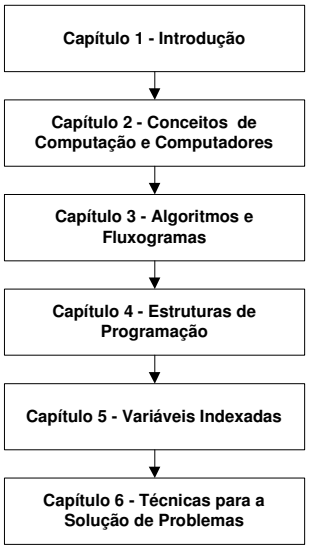
Por fim, no Capítulo 6 são discutidas as técnicas para a solução de problemas, mais especificamente as técnicas para modularizar a solução utilizando sub-rotinas. São apontados os dois tipos básicos de sub-rotinas (função e procedimento) e como empregá-los de acordo com a técnica *top-down* de modularização.

A Figura da página xxii exibe a organização dos capítulos deste livro.

Convenções tipográficas

Algumas convenções tipográficas foram utilizadas neste livro para tornar mais clara a sua compreensão:

- **Negrito** é empregado para destacar os conceitos importantes.
- *Itálico* é utilizado para enfatizar os conceitos essenciais e para palavras estrangeiras.



- Nos exercícios existem símbolos para identificar aqueles que são básicos, de resolução imediata; médios, nos quais o estudante deve pensar um pouco mais na solução; e desafios, a fim de empenhar-se mais na sua solução:
  - ☼ exercício fácil
  - 🌤 exercício médio
  - 🌧 exercício desafiador

Capítulo 1

Introdução

Um programa de computador é um produto resultante da atividade intelectual de um programador. Essa atividade, por sua vez, depende de um treinamento prévio em abstração e modelagem de problemas, bem como o uso da lógica na verificação das soluções. Neste capítulo são apresentados os conceitos introdutórios sobre a tarefa de programar computadores, a necessidade do uso de lógica na programação, a importância de se abstrair e modelar os problemas antes de partir para as soluções. Por fim, são apresentadas algumas dicas úteis que podem ser utilizadas na solução de problemas em geral.

1.1 O desenvolvimento de um software

Um **programa de computador** ou simplesmente **software** é representado pelas **instruções** e **dados** que algum ser humano definiu e que ao serem executados por alguma **máquina** cumprem algum **objetivo**. A máquina a que este texto se refere é um **computador digital**[1].

Os computadores digitais são máquinas eletrônicas contendo processadores e circuitos digitais adequados, operando com sinais elétricos em dois níveis ou **binários** (a ser detalhados no Capítulo 2).

Os dados são organizados em um computador de acordo com sua representação binária, isto é, seqüências de **0**s e **1**s. O objetivo de se utilizar um computador é extrair as **informações** resultantes de computações, isto é, o resultado das execuções das instruções de algum programa. Deve-se observar a diferença entre informação e dado: o dado

[1] Existem também computadores analógicos.

por si só é um valor qualquer armazenado em um computador enquanto a informação representa a *interpretação* desse dado, ou seja, qual o seu significado.

Parte dos dados processados durante a execução de um software é fornecida pelo ser humano (ou outra máquina) e denominada dados de **entrada**. Por outro lado, os dados de **saída** são aqueles fornecidos ao ser humano (ou outra máquina) após o processamento dos dados de entrada.

De qualquer forma, é importante notar que o objetivo do software é que motiva sua construção. Este pode ser definido como alguma necessidade humana, por exemplo, um programa para simular o funcionamento de um circuito digital, um programa para comandar um robô em uma linha de montagem, um sistema de gerenciamento de informações em uma empresa, somente para citar algumas. A Figura 1.1 descreve uma simplificação do processo de desenvolvimento de um software.

Figura 1.1 Simplificação do processo de construção de um software.

Nessa figura, o **cliente** especifica exatamente o que o software deve conter. Ele sabe **o que** o software deve conter e realizar, mas regra geral não sabe como. Ele indica o que o software deve contemplar e executar por meio de especificações chamadas **requisitos**.

Entende-se por cliente a entidade que contrata os serviços para a criação de um software, podendo ser uma empresa, pessoa ou ainda uma empresa que, por iniciativa própria, produza e venda seu software livremente (por exemplo, a Microsoft).

No desenvolvimento, os requisitos do cliente são traduzidos em **especificações técnicas** de software pelos **analistas de sistema** ou **engenheiros de software**. O desenvolvimento de um software é tipicamente dividido nas seguintes etapas:

- **Análise**: criam-se especificações que detalham como o software vai funcionar;
- **Projeto**: criam-se especificações que detalham o resultado da análise em termos mais próximos da implementação do software;

- **Implementação**: utilizando-se uma linguagem de programação e as especificações de projeto, o software é construído;
- **Testes**: após a construção do software, são realizados testes para conferir sua conformidade com os requisitos iniciais. O software deve satisfazer a todas especificações do cliente.

Por fim, após os testes o software é implantado na empresa. A implantação pode variar desde uma simples instalação via CD-ROM, que dure alguns minutos, até a instalação e testes de integração de diversos softwares, que pode levar semanas. De qualquer forma, o fato de o software estar finalizado e testado não significa que esteja totalmente livre de erros, também denominados *bugs*. Assim, deve-se voltar e tentar identificar a causa dos erros.

Pior que erros de programação, é o caso em que pode acontecer de o software funcionar corretamente, não apresentar erros, mas não realizar o que o cliente esperava. Nesse caso, deve-se retornar à etapa inicial, verificando os requisitos e refazendo todo o ciclo de desenvolvimento novamente.

É um fato que grande parte do investimento feito em um software é gasta na correção de erros do que propriamente na sua elaboração. Daí, surge a necessidade de enxergar o software como o produto de um processo bem-definido e controlado, que atue sobre as suas etapas de desenvolvimento, em outras palavras, um *processo de engenharia de software*.

1.2 Algoritmos e lógica de programação

Como foi brevemente apresentado na Seção 1.1, o software deve ser encarado como um produto de um processo bem-definido e controlado de engenharia. O intuito deste livro não é entrar em detalhes sobre engenharia de software e sim concentrar-se na disseminação de conceitos básicos que viabilizem a especificação correta de softwares, uma etapa imediatamente anterior a sua implementação ou programação.

O estudo de algoritmos e de lógica de programação é essencial no contexto do processo de criação de um software. Ele está diretamente relacionado com a etapa de projeto de um software em que, mesmo sem saber qual será a linguagem de programação a ser utilizada, se especifica completamente o software a ponto de na implementação ser possível traduzir diretamente essas especificações em linhas de código em alguma linguagem de programação como Pascal, C, Java e outras.

Essa tarefa permite verificar, em um nível maior de abstração, se o software está correto ou não. Permite, inclusive, averiguar se o software atenderá às especificações

originalmente propostas. Assim, evita-se partir diretamente para a etapa de implementação, o que poderá ocasionar mais erros no produto final.

1.2.1 O significado de um algoritmo

Um **algoritmo** representa um conjunto de regras para a solução de um problema. Essa é uma definição geral, podendo ser aplicada a qualquer circunstância que exija a descrição da solução. Dessa forma, uma receita de bolo é um exemplo de um algoritmo[2], pois descreve as regras necessárias para a conclusão de seu objetivo: a preparação de um bolo.

Em um bolo, descrevem-se quais serão os ingredientes e as suas quantidades. Depois, quais são as regras para o seu preparo, como a seqüência de inclusão dos ingredientes para bater as gemas; cozimento e assim por diante, conforme a Figura 1.2.

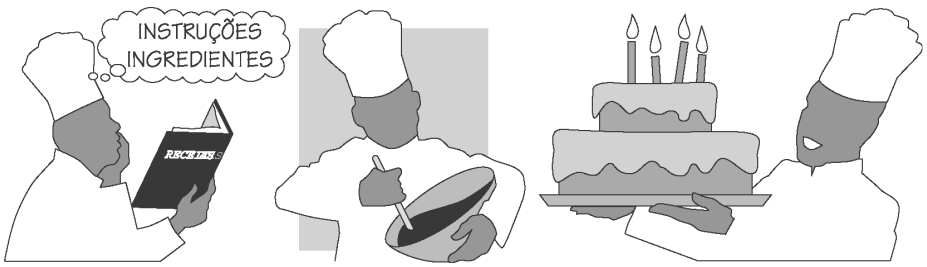


Figura 1.2 Uma receita de bolo é um algoritmo.

A correta execução das instruções contidas na receita de bolo leva à sua preparação. No entanto, se essas instruções tiverem sua ordem trocada ou a quantidade dos ingredientes alterada, o resultado vai divergir do original. Existe, ainda, o perigo de o autor da receita não a ter testado previamente, o que poderá gerar, novamente, resultados indesejáveis[3].

Da mesma forma, em programação, o algoritmo especifica com clareza e de forma correta as instruções que um software deverá conter para que, ao ser executado, forneça resultados esperados (veja a Figura 1.3).

[2] Na realidade, um algoritmo informal e impreciso.
[3] Se o fogão estiver desregulado, também vai gerar problemas.

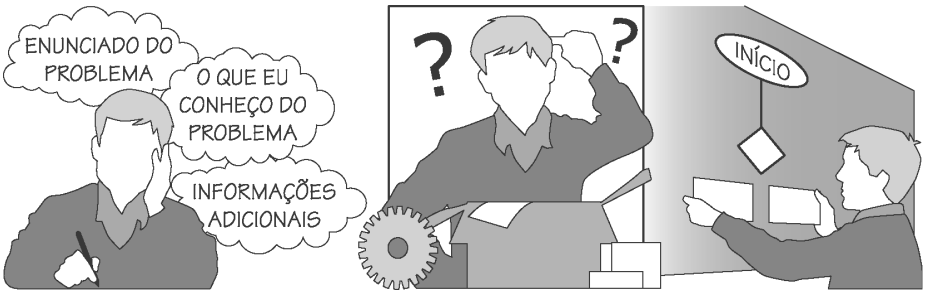


Figura 1.3 A tarefa de especificar um algoritmo.

Em primeiro lugar, deve-se saber qual é o problema a ser resolvido pelo software – o seu objetivo. Daí, deve-se extrair todas as informações a respeito desse problema (dados e operações), relacioná-las com o conhecimento atual que se tem do assunto, buscando eventualmente informações de outras fontes. Essa fase representa a **modelagem** do problema em questão e vai ser detalhada na Seção 1.4.2. A modelagem do problema é resultante de um processo mental de **abstração**, o qual será discutido na Seção 1.4.

Depois, sabendo como resolver o problema, a tarefa consiste em descrever claramente os passos para se chegar à sua solução. Os passos por si só não resolvem o problema; é necessário colocá-los em uma seqüência **lógica** (veja Seção 1.4.3), que, ao ser seguida, de fato o solucionará.

Além disso, é importante que essa descrição possua algum tipo de **convenção** para que todas as pessoas envolvidas na definição do algoritmo possam entendê-lo (veja a Seção 1.3). Chega-se, então, na especificação do algoritmo.

1.2.2 Exemplo de algoritmo

Considere o problema das *Torres de Hanoi*. A proposição do problema é a seguinte: inicialmente têm-se três hastes, A, B e C, e na haste A repousam três anéis de diâmetros diferentes, em ordem decrescente por diâmetro (veja a Figura 1.4).

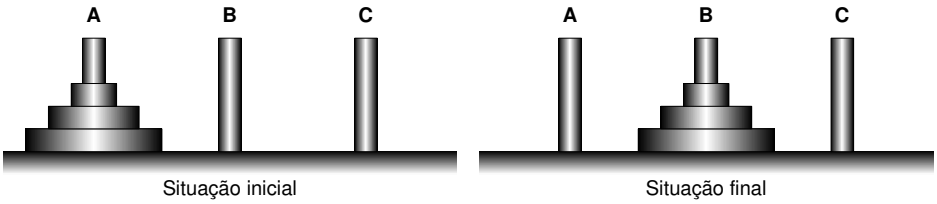


Figura 1.4 O problema das Torres de Hanoi.

O objetivo é transferir os três anéis da haste A para B, usando C se necessário. As regras de movimento são:

- deve-se mover um único anel por vez;
- um anel de diâmetro maior nunca pode repousar sobre algum outro de diâmetro menor.

As únicas informações para se resolver esse problema são as configurações inicial e final dos anéis e as regras de movimento. Uma solução poderia ser a seguinte seqüência de operações (veja o Algoritmo 1.1 e a Figura 1.5).

Algoritmo 1.1 Algoritmo para resolver o problema das Torres de Hanoi.

Início

1. Mover um anel da haste A para a haste B.
2. Mover um anel da haste A para a haste C.
3. Mover um anel da haste B para a haste C.
4. Mover um anel da haste A para a haste B.
5. Mover um anel da haste C para a haste A.
6. Mover um anel da haste C para a haste B.
7. Mover um anel da haste A para a haste B.

Fim

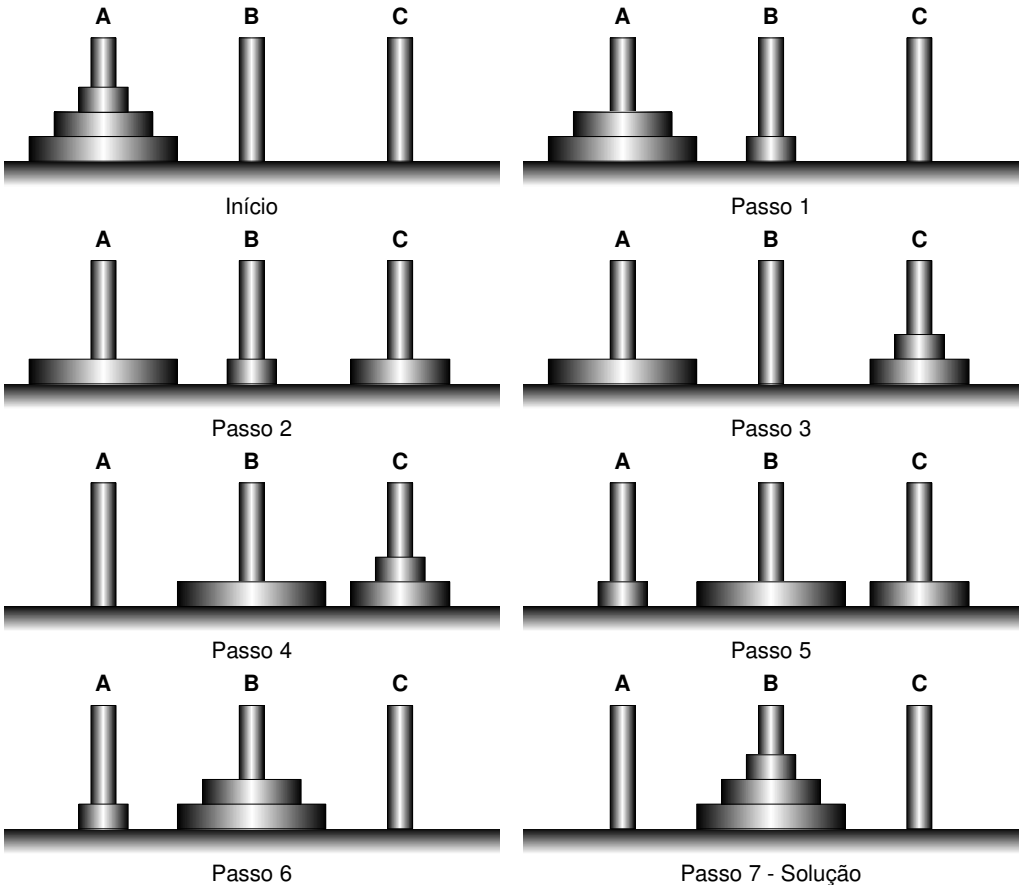


Figura 1.5 Solução do problema das Torres de Hanoi.

Como se obteve essa solução? Primeiro, é importante entender o enunciado do problema e as regras que foram impostas. Dessa forma, não é possível especificar os movimentos nos quais uma peça que esteja abaixo de outra seja movida e nem mover mais de uma peça por vez. Segundo, é importante verificar a cada passo definido se a solução está se aproximando do objetivo final.

O Algoritmo 1.2 define outra seqüência de operações para se solucionar o problema.

Algoritmo 1.2 Outro algoritmo para as Torres de Hanoi.

Início

1. Mover um anel da haste A para a haste C.
2. Mover um anel da haste A para a haste B.
3. Mover um anel da haste C para a haste B.
4. Mover um anel da haste A para a haste C.
5. Mover um anel da haste B para a haste C.
6. Mover um anel da haste B para a haste A.
7. Mover um anel da haste C para a haste A.
8. Mover um anel da haste C para a haste B.
9. Mover um anel da haste A para a haste C.
10. Mover um anel da haste A para a haste B.
11. Mover um anel da haste C para a haste B.

Fim

Deve-se observar que essa solução é válida e também resolve o caso das Torres de Hanoi. No entanto, leva-se mais tempo para chegar se à solução (onze passos contra sete da solução anterior). Esse exemplo demonstra que uma mesma solução pode ser melhor ou pior que outra. Isso é um conceito importante quando se trata de um programa, pois, dependendo do problema, determinar uma solução mais eficiente pode economizar até horas de processamento. Outras soluções válidas também podem ser propostas, mas não terão menos que sete movimentos.

Agora, e se for considerado o mesmo problema, porém, com n anéis postados inicialmente na haste A? Como isso afetará a solução? É interessante estudar diversos casos particulares antes de elaborar uma solução genérica.