

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, sequências de **0s** e **1s**. 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

¹Existem 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, 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 algoritmo², 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 sequência de inclusão dos ingredientes para bater as gemas; cozimento e assim por diante, conforme a Figura 1.2.

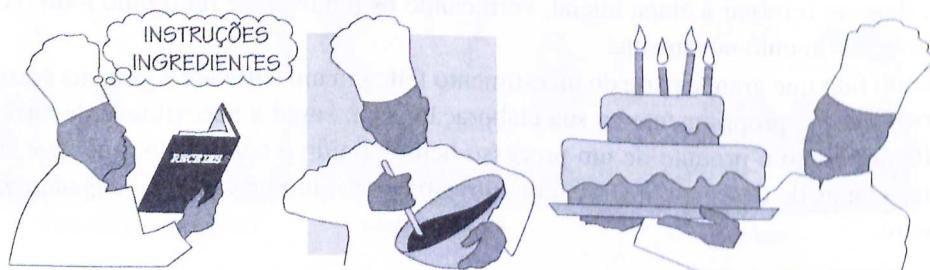


Figura 1.2 Uma receita de bolo é um algoritmo.

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 indesejáveis³.

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).

²Na realidade, um algoritmo informal e impreciso.

³Se o fogão estiver desregulado, também vai gerar problemas.

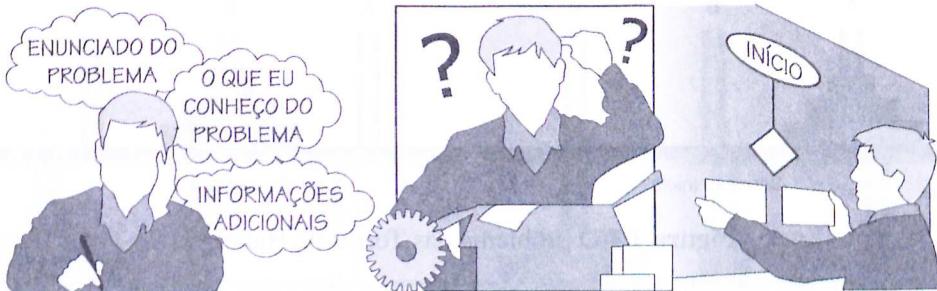


Figura 1.3 A tarefa de especificar um algoritmo.

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 sequência **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 hastas, *A*, *B* e *C*, e na haste *A* repousam três anéis de diâmetros diferentes, em ordem decrescente por diâmetro (veja a Figura 1.4).

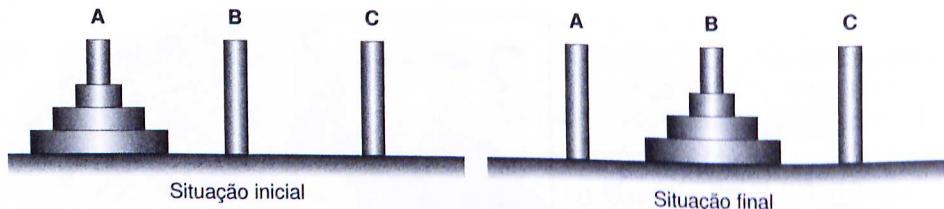


Figura 1.4 O problema das Torres de Hanoi.

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 sequência 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

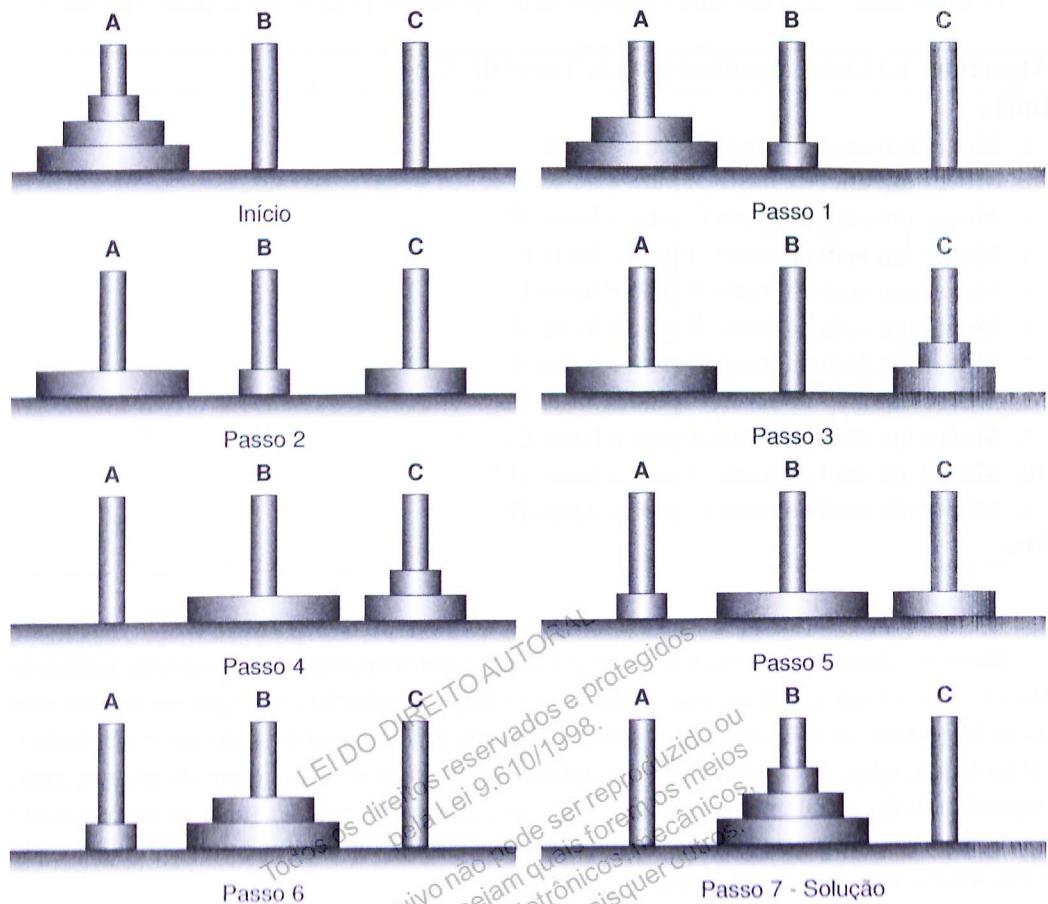


Figura 1.5 Solução do problema das Torres de Hanoi.

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 sequência 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 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.

A Tabela 1.1 demonstra alguns resultados do problema das Torres de Hanoi quando se varia n (considere que $X \rightarrow Y$ representa a operação “mover uma peça da haste X para Y ”).

Tabela 1.1 Alguns resultados para o problema das Torres de Hanoi.

n	Número de movimentos	Movimentos
1	1	$A \rightarrow B$
2	3	$A \rightarrow C; A \rightarrow B; C \rightarrow B$
3	7	$A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; C \rightarrow B; A \rightarrow B$
4	15	$A \rightarrow C; A \rightarrow B; C \rightarrow B; A \rightarrow C; B \rightarrow A; B \rightarrow C; A \rightarrow C; A \rightarrow B; C \rightarrow B; C \rightarrow A; B \rightarrow A; C \rightarrow B; A \rightarrow C; A \rightarrow B; C \rightarrow B$
5	31	$A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; C \rightarrow B; A \rightarrow B; A \rightarrow C; B \rightarrow C; B \rightarrow A; C \rightarrow A; B \rightarrow C; A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; B \rightarrow C; B \rightarrow A; C \rightarrow A; C \rightarrow B; A \rightarrow B; A \rightarrow C; B \rightarrow C; A \rightarrow B; C \rightarrow A; C \rightarrow B; A \rightarrow B$

É possível notar pela Tabela 1.1 que existe uma relação matemática entre n e o número de anéis do problema com o número de movimentos executados, sendo a relação $m(n) = 2^n - 1$, em que m representa os números de movimentos.

Assim, se n for igual a 20, o número de movimentos que deverão ser descritos será igual a 1.048.575 (haja lápis e papel!). Portanto, a generalização do algoritmo desse problema, na forma em que está sendo descrito, é impraticável para grandes valores de n . É necessário determinar formas mais sintéticas para expressar a solução desse estudo de caso.

Um modo de expressar a solução geral desse problema de forma mais sintética é enxergá-lo de outra maneira. Considere que as três hastes estão dispostas em um círculo, conforme ilustrado na Figura 1.6.

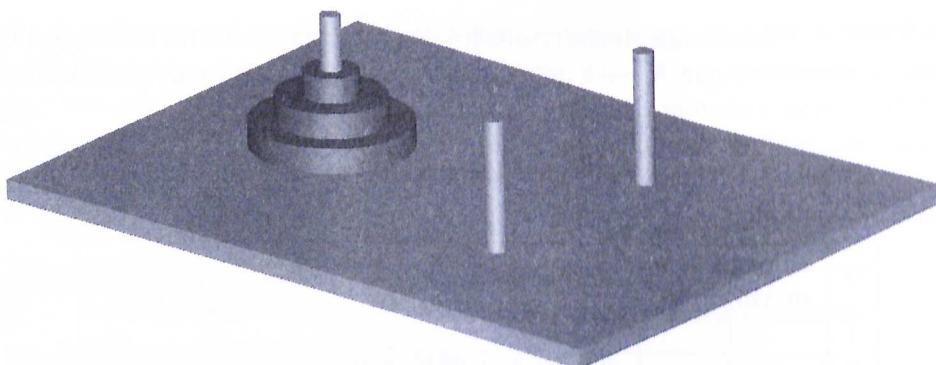


Figura 1.6 Preparação para o uso do algoritmo geral para as Torres de Hanoi.

Um algoritmo que proporciona uma solução geral para o problema das Torres de Hanoi é descrito pelo Algoritmo 1.3. Esse algoritmo não impõe qual deve ser a haste destino. Observa-se que para os valores de n ímpar, os anéis serão transferidos para a primeira haste que estiver após a haste de início, no sentido horário. Se n for par, os anéis serão transferidos para a primeira haste que estiver após aquela de início, no sentido anti-horário.

Algoritmo 1.3 Algoritmo geral para as Torres de Hanoi.

Início

1. **Repita** {repetir a execução das duas linhas abaixo até que a condição na parte **até** seja atendida.}
2. Mova o menor anel de sua haste atual para a próxima, no sentido horário.
3. Execute o único movimento possível com um anel que não seja o menor de todos.
4. **Até** que todos os discos tenham sido transferidos para outra haste.

Fim

Todos os direitos reservados e protegidos
pela Lei 9.609/98.
Este arquivo não pode ser
transmitido sejam quais forem os meios
empregados: eletrônicos, mecânicos,
fotográficos ou quaisquer outros.

A Figura 1.7 mostra a aplicação desse algoritmo para n valendo 3.

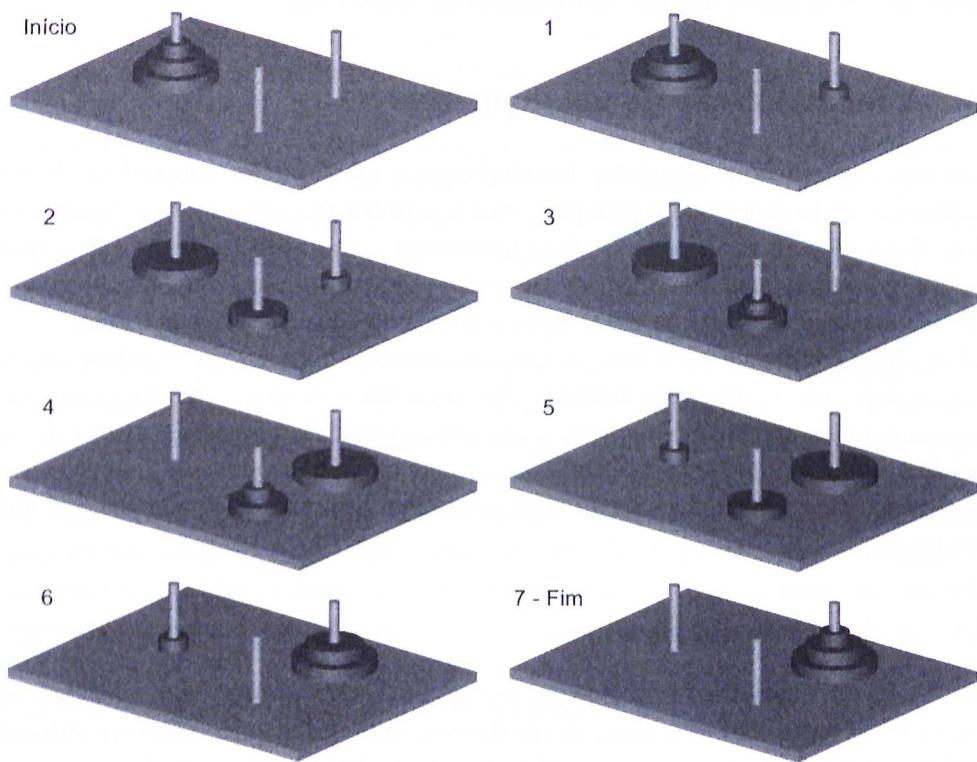


Figura 1.7 Uso do algoritmo geral para o problema das Torres de Hanoi.

Essa solução funciona para qualquer valor de n tal que $n \geq 1$. Apesar de haver um algoritmo geral que resolve todos os casos para o problema das Torres de Hanoi, ainda existem algumas dificuldades na sua forma de descrição:

- Existe um tratamento informal dos comandos: o que significa *repita e até*. É necessário formalizar os comandos de algoritmo.
- As operações descritas pelo algoritmo são úteis para uma pessoa interpretar, mas não para uma máquina. É necessário modelar melhor o problema de forma que este seja facilmente traduzido para uma máquina. Está implícito, por exemplo, que não deve ser realizado nenhum movimento no passo 3 se não existir nenhuma outra peça que possa ser movida além do menor anel.

A necessidade de se formalizar um algoritmo é discutida na Seção 1.3 e os tópicos de modelagem de problemas são apresentados na Seção 1.4.2.

1.3 A formalização de um algoritmo

A tarefa de especificar os algoritmos para representar um programa consiste em detalhar os dados que serão processados pelo programa e as instruções que vão operar sobre esses dados. Essa especificação pode ser feita livremente como visto na Seção 1.2.2, mas é importante **formalizar** a descrição dos algoritmos segundo alguma **convenção**, para que todas as pessoas envolvidas na sua criação possam entendê-lo da mesma forma.

Em primeiro lugar, é necessário definir um conjunto de regras que regulem a escrita do algoritmo, isto é, regras de **sintaxe**. Em segundo, é preciso estabelecer as regras que permitam interpretar um algoritmo, que são as regras **semânticas**. Apesar de essa formalização ser assunto para os Capítulos 3 e 4, nesta seção são resumidos os conceitos importantes sobre a formalização de algoritmos com o intuito de já ambientar o leitor nesse tópico.

1.3.1 A sintaxe de um algoritmo

A sintaxe de um algoritmo resume-se nas regras para escrevê-lo corretamente. Em computação, essas regras indicam quais são os tipos de **comandos** que podem ser utilizados e também como neles escrever **expressões**. As expressões de um comando em um algoritmo realizam algum tipo de **operação** com os **dados** envolvidos, isto é, operam com valores e resultam em outros valores que são usados pelo algoritmo.

Os tipos de comandos de um algoritmo são também denominados **estruturas de programação**. Existem apenas três tipos de estruturas que podem ser utilizadas para escrever qualquer programa: **estruturas sequenciais, de decisão e de repetição**. Por exemplo, o Algoritmo 1.4 emprega apenas as estruturas sequenciais, pois sua execução é direta, imperativa, não havendo nenhum tipo de condição a ser verificada e nenhum desvio em seu caminho. Já o Algoritmo 1.3 usa uma estrutura de repetição, que possui uma condição que, se for verdadeira, terminará sua execução. Enfatiza-se novamente a vantagem de se ter utilizado essa estrutura, uma vez que ela evita que se tenha de escrever todos os $2^n - 1$ comandos de movimentação dos anéis para um problema da Torre de Hanoi com $n \geq 1$.

As expressões que são escritas em estruturas de programação envolvem a utilização de dados. Antecipando o que será visto no Capítulo 2, os dados em um computador são números binários, isto é, sequências de 0s e 1s, e são armazenados em sua memória.

Não é prático trabalhar diretamente com essa representação e então convencionou-se que os dados manipulados por um programa são categorizados em **tipos de dados**, que torna simples seu uso para o programador, mas que na realidade, para a máquina, são traduzidos em valores binários. Assim é possível manipular diversos tipos de dados em um algoritmo: números inteiros e reais, valores lógicos, textos etc.

A manipulação desses dados é feita por meio de **variáveis** e **valores constantes**, que representam no texto do algoritmo os dados que serão armazenados na memória do computador. O significado de variável é similar àquele empregado na matemática: representar um valor, porém com um significado físico por trás; esse valor será armazenado na memória de um computador. Um valor constante representa um valor que não pode ser alterado, como o número 25, o nome ‘Márcio’ e assim por diante.

Uma variável pode ser manipulada de muitas formas. Os valores constantes ou resultados de expressões envolvendo as variáveis podem ser **atribuídos** a estas. Para escrever as expressões corretamente, é necessária também uma sintaxe. Essa sintaxe depende do tipo de variáveis envolvidas e determina quais são os **operadores** que podem ser aplicados. Além dos operadores propriamente ditos, especificam-se algumas **funções** e **procedimentos** predefinidos úteis, que simplificam algumas tarefas corriqueiras em computação, como ler os dados digitados por um usuário, escrever resultados na tela do computador, calcular o seno de um número etc.

1.3.2 Exemplo de sintaxe de um algoritmo

Deseja-se especificar um algoritmo para calcular e exibir na tela a área de um triângulo de base b e altura h , em que os valores de b e de h são fornecidos pelo usuário via teclado. Antes de mais nada, a solução desse problema é imediata, pois sabe-se que a área s de um triângulo de base b e altura h é dada por $s = \frac{b \times h}{2}$.

Um algoritmo para se resolver esse problema pode ser definido de maneira informal, como ilustrado pelo Algoritmo 1.4. Observe que essa descrição está em português e não é conveniente que seja traduzida para uma linguagem de computador.

Algoritmo 1.4 Algoritmo informal para calcular a área de um triângulo.

Início

1. Pedir para o usuário digitar os valores de b e de h .
2. Calcular a área s usando a fórmula $s = \frac{b \times h}{2}$.
3. Exibir o valor de s na tela.

Fim

Agora, a tarefa é descrever esse mesmo algoritmo, utilizando alguma representação mais formal. Primeiro, será considerada a representação por fluxograma (veja Capítulo 3) cujo resultado é apresentado na Figura 1.8.

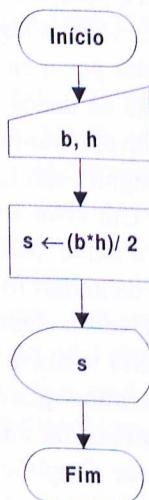


Figura 1.8 Fluxograma para calcular a área de um triângulo.

Nessa representação, a sintaxe para a escrita de um algoritmo é dada pelos **símbolos** do fluxograma e pelas regras para a escrita das expressões. A regra geral de um fluxograma estabelece que este deva ser escrito com seus símbolos básicos, interligados por linhas com ou sem setas que indicam a direção em que os comandos devem ser executados. Sua interpretação deve começar em um símbolo de *início* e terminar em um símbolo de *fim*. O *início* e *fim* do algoritmo são representados por dois retângulos de cantos arredondados, que são sempre os mesmos para qualquer fluxograma.

Os dados do problema – a base, a altura e a área – são representados pelas variáveis *b*, *h* e *s*. O símbolo do trapézio representa um comando que possibilita ao usuário digitar os valores que serão atribuídos às variáveis *b* e *h*. O símbolo do retângulo representa um comando a ser executado de forma imperativa.

Por sua vez, o comando representado pela flecha esquerda (\leftarrow) permite copiar para a variável *s* o valor da expressão $(b * h) / 2$. Observe que existe uma regra para se escrever essa expressão: * significa multiplicação e /, divisão. Por fim, o símbolo de um retângulo de cantos arredondados mais achatado à esquerda significa exibir o valor da variável *s* na tela.

Agora será considerado esse mesmo algoritmo, utilizando-se uma representação em pseudocódigo denominado *Portugol* (veja o Capítulo 4), apresentado pelo Algoritmo 1.5.

Algoritmo 1.5 Algoritmo em Portugol para calcular a área de um triângulo.

Início

1. **Leia**(b, h).
2. $s \leftarrow (b * h) / 2$.
3. **Exiba**(s).

Fim

Nesse caso, a sintaxe é dada pelos comandos dessa pseudolínguagem. A regra geral para se escrever um algoritmo nessa representação determina que este deva ser delimitado pelas palavras *início* e *fim* e os comandos, logo após o símbolo de início, devem ser executados sequencialmente, de cima para baixo. O comando *leia* corresponde a um procedimento que automaticamente pede para o usuário digitar dois valores e estes são copiados para as variáveis b e h . A expressão do cálculo de s tem o mesmo significado que no caso do fluxograma, e o comando *exiba* mostra na tela do computador o valor da variável s .

1.3.3 A semântica de um algoritmo

Como já foi exposto, a semântica de um algoritmo estabelece regras para sua interpretação. Os símbolos ou comandos de um algoritmo por si só não têm um significado, a menos que este seja bem-definido.

Por exemplo, no caso de um fluxograma, o símbolo de retângulo representa um comando que deve ser executado de forma imperativa, isto é, sem condição alguma. É definido geralmente como um símbolo e, no seu interior, figura uma expressão que deve ser avaliada, podendo ser qualquer uma desde que seja válida para esse símbolo.

No exemplo do fluxograma da Figura 1.8, a expressão utilizada no interior do retângulo é $s \leftarrow (b * h) / 2$, caracterizando-se por ser válida, pois usa os símbolos definidos para um fluxograma, e sua semântica é *multiplicar o valor de b pelo valor de h, dividir esse resultado por 2 e copiarlo para a variável s*.

Dessa forma, a semântica de um algoritmo sempre acompanha a sua sintaxe, fornecendo um significado. A importância da formalização de um algoritmo, sua sintaxe e semântica podem ser resumidos assim:

- Evitar ambiguidades, pois definem regras sintáticas e semânticas que sempre são interpretadas da mesma forma.
- Impedir a criação de símbolos ou comandos desnecessários na criação de um algoritmo: representam um conjunto mínimo de regras que pode ser utilizado em qualquer algoritmo.
- Permitir uma aproximação com as regras de uma linguagem de programação, fazendo, assim, uma fácil tradução de um algoritmo para sua implementação no computador.

1.4 Como resolver problemas

A criação de um algoritmo é uma tarefa essencialmente intelectual. A partir do enunciado de um problema, deseja-se obter um algoritmo que o resolva. Pode-se afirmar que a tarefa de escrever algoritmos é, portanto, uma tarefa de resolver problemas.

1.4.1 A análise e a síntese de um problema

A resolução de um problema envolve duas grandes fases: a **análise** e a **síntese** da solução.

Na fase de análise, o problema é entendido de forma que se descubra o que deve ser solucionado, quais são os dados necessários e as condições para resolvê-lo se esses dados e essas condições são necessários, insuficientes ou redundantes ou ainda contraditórios e, então, parte-se para a sua **modelagem**, podendo ser enriquecida com o auxílio de equações, desenhos ou gráficos. Como resultado dessa fase, tem-se a elaboração de um **plano de ação**, no qual a experiência em problemas similares vistos anteriormente é utilizada e, também, pode ser necessária a utilização de problemas auxiliares. Nessa fase, faz-se uso direto de processos de **abstração**, o que significa elaborar modelos mentais do problema em questão e do encaminhamento de sua solução.

Na etapa de síntese, executa-se o plano definido na fase de análise, representando os passos por meio de um algoritmo. Aqui, emprega-se uma representação formal, como visto na Seção 1.3. É importante que a solução seja verificada e comprovada corretamente, por meio da execução do algoritmo. Essa execução é feita percorrendo-se o algoritmo do seu início até o seu final, e verificando, a cada passo, se o resultado esperado foi obtido. Caso tenha sido encontrada alguma discrepância, deve-se procurar saber qual foi sua causa e eventualmente analisar novamente o problema, repetindo-se assim esse ciclo até que a solução tenha sido obtida.

1.4.2 Modelagem de problemas

A modelagem (geralmente desprezada) é a principal responsável pela facilidade ou dificuldade da resolução de um problema. Na Matemática e na Engenharia, por exemplo, o uso da **linguagem matemática** é fundamental, principalmente pela eliminação de duplos sentidos que acontecem nessa prática.

O mesmo ocorre na computação, com o emprego de linguagens de descrição de algoritmos (como fluxogramas) e de linguagens de programação (como a linguagem Pascal).

Como um exemplo de modelagem, considere o seguinte problema:

Compraram-se 30 canetas iguais, que foram pagas com uma nota de R\$ 100,00, obtendo-se R\$ 67,00 como troco. Quanto custou cada caneta?

Este é um problema bem simples, cuja solução pode até ser feita “de cabeça”; porém, como pode ser mostrada a solução? Uma possível resposta:

Se eu tinha R\$ 100,00 e recebi como troco R\$ 67,00, o custo do total de canetas é a diferença entre os R\$ 100,00 que eu tinha e os R\$ 67,00 do troco. Ora, isto vale R\$ 33,00; portanto, esse valor foi o total pago pelas canetas. Para saber quanto custou cada caneta, basta dividir os R\$ 33,00 por 30, resultando no preço de cada caneta. Assim, cada caneta custou o equivalente a R\$ 1,10.

Esse raciocínio é matematicamente demonstrado por: seja x o custo de cada caneta, então $quantogastei = 30x$. Como $quantogastei + \text{troco} = R\$ 100,00$, tem-se:

$$\begin{aligned} 30x + 67 &= 100 \\ 30x &= 100 - 67 \\ 30x &= 33 \\ x &= \frac{33}{30} \\ x &= 1,1 \end{aligned}$$

□

De uma forma mais curta e universalmente entendida, pode-se também dizer que o caminho pode ser obtido por um algoritmo como o Algoritmo 1.6.

Algoritmo 1.6 Algoritmo inicial para solucionar o problema das canetas.**Início**

1. Pegar os valores 30, 100 e 67.
2. Subtrair 67 de 100 e dividir o resultado por 30.
3. Mostrar o resultado final.

Fim

Deve-se observar que esse algoritmo resolve apenas uma **instância particular do problema** das canetas. E para solucionar um **caso geral**, tem-se o seguinte:

Compraram-se N canetas iguais, que foram pagas com uma nota de Z reais, obtendo-se Y reais como troco. Quanto custou cada caneta?

Na solução desse problema mais geral, utiliza-se a experiência que foi adquirida no problema particular e sintetizada pelo Algoritmo 1.6. Nesse caso, basta que sejam fornecidos os valores das variáveis N , Z e Y que a solução do problema é a mesma que a anterior e seu algoritmo pode ser descrito como no Algoritmo 1.7.

Algoritmo 1.7 Algoritmo geral para solucionar o problema das canetas.**Início**

1. Ler os valores de N , Y e Z .
2. Subtrair Y de Z e dividir o resultado por N .
3. Mostrar o resultado final.

Fim

No entanto, a proposta apresentada tem uma série de restrições: o que acontecerá se alguém pensar em comprar zero canetas? Ou 3 canetas faz algum sentido? Para alguém que não possua a forma de interpretar os resultados, isto é possível. Suponha que alguém execute o Algoritmo 1.7 com os seguintes valores: $N = 10$, $Z = 10$ e $Y = 15$. O valor de cada caneta será de -R\$ 0,50. Isto faz algum sentido?

Então é necessário que, para a solução geral apresentada pelo Algoritmo 1.7 ser realmente consistente, seja levada em consideração a precondição do problema em que o valor pago pelas canetas seja sempre maior que o troco recebido, que o valor pago e a quantidade de canetas seja sempre maior que zero e que o troco seja maior ou igual a zero.

Em termos matemáticos, podem-se expressar essas condições como $Z > Y$, $N > 0$, $Z > 0$ e $Y \geq 0$. Se essas condições forem todas verdadeiras, então, aplica-se a fórmula conhecida e obtém-se o resultado. Caso contrário, o algoritmo deve terminar sinalizando,

de alguma forma, a razão de seu fracasso. O algoritmo correto para a solução geral das canetas está apresentado pelo Algoritmo 1.8.

Algoritmo 1.8 Algoritmo geral e correto para solucionar o problema das canetas.

Início

1. Ler os valores de N , Y e Z .
2. **Se** $Z > Y$ e $N > 0$ e $Y \geq 0$ e $Z > 0$ **Então**
3. Subtrair Y de Z e dividir o resultado por N .
4. Mostrar o resultado final.
5. **Senão**
6. Exibir a mensagem: “Erro: os valores são inconsistentes!”.
7. **Fim Se**

Fim

Essa solução pode ser agora formalizada. Utilizando-se a notação de fluxogramas, ela poderia ser representada como exposta na Figura 1.9. É possível descobrir qual a função de todos os símbolos desse fluxograma ao compará-lo com o Algoritmo 1.8?

1.4.3 O papel da lógica em programação

Lógica é uma área da Matemática cujo objetivo é investigar a veracidade de suas proposições. Considere, por exemplo, o caso em que temos as seguintes proposições:

1. Se estiver chovendo, eu pegarei meu guarda-chuva.
2. Está chovendo.

O que se conclui dessas duas proposições? Parece que a conclusão óbvia é: “eu pegarei meu guarda-chuva”.

Essa conclusão segue o fato de que existe uma *implicação lógica* na primeira proposição, a qual afirma que “se estiver chovendo” implica “eu pegarei meu guarda-chuva”. Essa implicação age como uma *regrá*⁴, que conduz à dedução do fato. Continuando, e se as proposições fossem:

1. Se estiver chovendo, eu pegarei meu guarda-chuva.
2. Eu peguei meu guarda-chuva.

O que se conclui? A conclusão poderia ser: “é plausível que esteja chovendo”⁴.

⁴Não se pode afirmar com precisão que está chovendo só porque você pegou seu guarda-chuva!

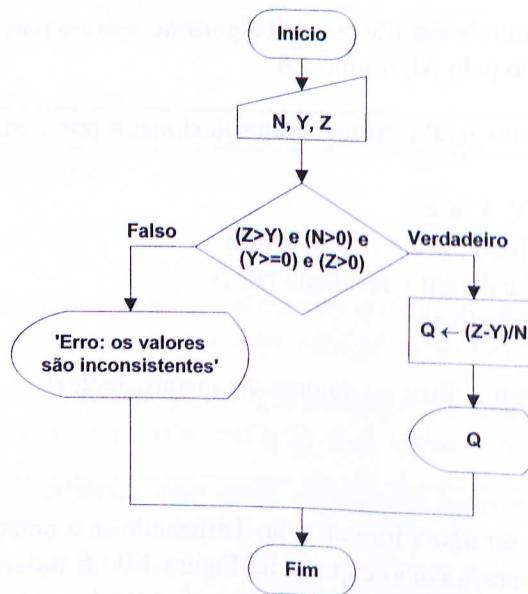


Figura 1.9 Fluxograma para resolver o problema das canetas.

Esses dois exemplos fornecem uma ideia, embora simplificada, do que a lógica se preocupa em estudar. Toda lógica proposta também deve ser formalizada em *elementos sint ticos* (especificam como escrever suas proposi es) e *elementos sem ticos* (avaliam o significado das proposi es – suas interpreta es). No caso da l gica cl ssica, o resultado da avalia o de suas proposi es pode ser somente um entre dois valores: **VERDADEIRO** ou **FALSO**. N o se entrará em mais detalhes sobre essa formaliza o, pois esse assunto est  al m do escopo deste livro.

O papel da l gica em programaci o de computadores est  relacionado com a correta sequ ncia de instru es que devem ser definidas para que o programa atinja seu objetivo. Serve como instrumento para a verifica o do programa escrito, provando se este est  correto ou n o.

Em um algoritmo em execuci o, o valor das suas vari veis a cada instante representa o seu **estado**. Com a execuci o dessas instru es, esse estado vai sendo alterado. Um algoritmo **correto**  e aquele que, a partir de um **estado inicial** de suas vari veis, consegue, com a execuci o de suas instru es, chegar a um **estado final**, no qual os valores das vari veis est o de acordo com a solu o esperada.

Voltando-se ao algoritmo geral para a solu o do problema das canetas, podemos conferir sua solu o analisando a l gica empregada em cada passo. Isso pode ser confe-

rido pelo Algoritmo 1.9, que possui alguns comentários nas instruções.

Algoritmo 1.9 Algoritmo correto, comentado, para solucionar o problema das canetas.

Início

1. Ler os valores de N , Y e Z . {Nesse ponto temos três valores quaisquer de N , Y e Z .}
2. Se $Z > Y$ e $N > 0$ e $Y \geq 0$ e $Z > 0$ Então {Nesse ponto, garante-se que $Z > Y$, $N > 0$, $Y \geq 0$ e $Z > 0$ }
3. Subtrair Y de Z e dividir o resultado por N . {Como $Z > Y$, $Y \geq 0$ e $Z > 0$ então $Z - Y > 0$ e sendo $N > 0$, o resultado existe e é maior que zero.}
4. Mostrar o resultado final. {É exibido o resultado, que existe e é maior que zero.}
5. Senão
6. Exibir a mensagem: “Erro: os valores são inconsistentes!”. {Nesse ponto, pelo menos uma das condições do problema não foi atendida.}

7. Fim Se

Fim

No Algoritmo 1.9 provou-se, portanto, que a sua lógica está correta e que leva a resultados esperados. Se forem digitados os valores que atendam às condições do algoritmo, será calculado um valor. Caso contrário, será exibida uma mensagem de erro.

Este é, por conseguinte, o papel da lógica na programação: provar que um algoritmo que foi elaborado está correto. Juntamente com a simulação do algoritmo, que consiste em executá-lo com dados reais, é possível saber se está correto e se leva a valores consistentes.

1.5 Como se portar em um curso de computação

O grande problema apresentado pelos estudantes em um primeiro curso de computação não são as linguagens de programação ou de descrição de algoritmos propriamente ditas, mas sim a dificuldade em abstrair e descrever as soluções de problemas contando apenas com poucas e simples estruturas.

O que deve ser percebido é que o sucesso em um curso ou carreira de computação exige uma predisposição em se envolver com tarefas diretamente intelectuais. Um novo problema de computação pode ser gerado a partir de um já existente, alterando-se apenas poucos elementos de seu enunciado. Isso é facilmente percebido pelos exemplos do problema das Torres de Hanoi (Seção 1.2.2) e pelo problema das canetas (Seção 1.4.2).

Dessa forma, é um **erro decorar** as soluções em computação, pois podem não servir

para outros problemas, que com certeza serão diferentes. O que deve ser procurado é o **entendimento** de como foi obtida uma solução, armazená-la na memória⁵ e então utilizar essa experiência adaptando-a a outras situações, por **analogia, generalização ou especialização**. A grande dica é que um problema pode ser diferente de outro, mas consegue-se aproveitar grande parte da experiência obtida em problemas passados em novos desafios.

Não existe em computação uma “fórmula mágica” para resolver problemas. De qualquer forma, apresenta-se a seguir um conjunto de dicas que podem ser utilizadas durante o processo de raciocínio empregado na resolução de problemas:

1. Ao se deparar com um problema novo, tente entendê-lo. Para auxiliar, pense no seguinte:

- O que se deve descobrir ou calcular? Qual é o objetivo?
- Quais são os dados disponíveis? São suficientes?
- Quais as condições necessárias e suficientes para resolver o problema?
- Faça um esboço informal de como ligar os dados com as condições.
- Se possível, modele o problema de forma matemática.

2. Crie um plano com a solução:

- Consulte sua memória e verifique se você já resolveu algum problema similar. A sua solução pode ser aproveitada por *analogia*, quando o enunciado for diferente, mas a estrutura em si guarda similaridades; por *generalização*, quando se tem uma solução particular e deseja uma solução geral; por *especialização*, quando se conhece alguma solução geral que serve como base para uma em particular ou ainda uma mistura das três técnicas anteriores.
- Verifique se é necessário introduzir algum elemento novo no problema, como um problema auxiliar.
- Se o problema for muito complicado, tente quebrá-lo em partes menores e solucionar essas partes.
- É possível enxergar o problema de outra forma, de modo que seu entendimento se torne mais simples?

3. Formalize a solução:

- Crie um algoritmo informal com passos que resolvam o problema.

⁵A sua e não a do computador.

- Verifique se cada passo desse algoritmo está correto.
- Escreva um algoritmo formalizado por meio de um fluxograma ou outra técnica de representação.

4. Exame dos resultados:

- Teste o algoritmo com diversos dados de entrada e verifique os resultados (teste de mesa).
- Se o algoritmo não gerou resultado algum, o problema está na sua sintaxe e nos comandos utilizados. Volte e tente encontrar o erro.
- Se o algoritmo gerou resultados, estes estão corretos? Analise sua consistência.
- Se não estão corretos, alguma condição, operação ou ordem das operações está incorreta. Volte e tente encontrar o erro.

5. Otimização da solução:

- É possível melhorar o algoritmo?
- É possível reduzir o número de passos ou dados?
- É possível conseguir uma solução ótima?

Finalizando este capítulo, os problemas de computação são verdadeiros projetos de Engenharia. Aqui se tem a oportunidade de analisar um problema, definir uma estratégia de solução e, por fim, criar um produto, que é o programa em si.

Os programas de computador obtidos não devem ser vistos como objetos isolados do mundo e sim como ferramentas que são empregadas para auxiliar diversas áreas da atividade humana. Dessa forma, este livro utiliza uma abordagem multidisciplinar, na qual os conceitos de outras disciplinas da Engenharia como cálculo, geometria analítica, física etc., são usados nos enunciados dos exercícios, e o objetivo é resolver problemas pertinentes à Engenharia, utilizando o computador como sua ferramenta de trabalho.

1.6 Exercícios

Segue um conjunto de exercícios de lógica que necessitam apenas de raciocínio e bom senso para serem resolvidos. Tente resolvê-los à sua maneira, lembrando-se do que foi discutido neste capítulo. O modo de resolução é livre, mas podem ser utilizadas equações ou frases em português. Faça do seu jeito.

- 1.1.** ☀ Descreva como descobrir a moeda falsa em um grupo de cinco moedas, fazendo uso de uma balança analítica (sabe-se que a moeda falsa é mais leve que as outras), com o menor número de pesagens possível. Lembre-se de que sua descrição deve resolver o problema para qualquer situação.

Dica: É possível resolver com apenas duas pesagens.

- 1.2.** ☀ Idem ao anterior, porém só se sabe que a moeda falsa tem massa diferente. Para descobrir se ela é mais leve ou mais pesada que as outras, muda-se alguma coisa?

- 1.3.** ☀ Idem ao Exercício 1.1, porém com nove moedas.

- 1.4.** ☀ Têm-se três garrafas, com formatos diferentes, uma cheia até a boca, com capacidade de oito litros e as outras duas vazias com capacidades de cinco e três litros, respectivamente. Deseja-se separar o conteúdo da primeira garrafa em duas quantidades iguais. Elabore uma rotina que consiga realizar a tarefa, sem que se possa fazer medidas.

- 1.5.** ☀ Um caramujo está na parede de um poço a cinco metros de sua borda. Tentando sair do poço, ele sobe três metros durante o dia, porém desce escorregando dois metros durante a noite. Quantos dias levará para o caramujo conseguir sair do poço?

- 1.6.** ☀ Um tijolo “pesa” um quilo mais meio tijolo. Quanto quilos “pesam” um tijolo e meio?

- 1.7.** ☀ Você está em uma margem de um rio, com três animais: uma galinha, um cachorro e uma raposa. Somente pode atravessar com um animal por vez e nunca deixar a raposa e o cachorro sozinhos nem a raposa e a galinha. Descreva uma forma de conseguir atravessar os três animais, obedecendo a essas condições.

- 1.8.** ☀ Você dispõe de uma balança precisa e dez sacos cheios de moedas idênticas na aparência, das quais todas as moedas de um dos sacos são falsas e de massa 1 g menor que as verdadeiras. Qual o menor número de pesagens necessárias para se descobrir o saco de moedas falsas?

1.9. ☀ A prova de que $2 = 1$. Considere $a = b = 1$:

$$\begin{aligned} a &= b \\ ab &= b^2 \\ ab - a^2 &= b^2 - a^2 \\ a(b - a) &= (b + a)(b - a) \\ a &= \frac{(b + a)(b - a)}{(b - a)} \\ a &= b + a \\ 1 &= 2 \quad \square \end{aligned}$$

A matemática que você estudou até agora é válida? Então, onde está o erro na dedução anterior?

1.10. ☀ Considere o Algoritmo 1.10.

Algoritmo 1.10 Algoritmo para o Exercício 1.10.

Início

1. Ler os valores de A e B
2. $C \leftarrow 0$
3. **Enquanto** $A > B$ **Faça**
4. Subtraia B de A , coloque o resultado em A e some 1 em C
5. **Fim Enquanto**
6. Mostre os valores finais de C e de A

Fim

Execute essas instruções para os seguintes pares de números: 10 e 2, 6 e 2, 15 e 3. O que significa o valor final de C ? E o valor final de A ?

1.11. ☀ Dois amigos se encontraram em uma rua. Eles não se viam há alguns anos. Um dos amigos, aproveitando que o outro é um professor de Matemática, inicia o seguinte diálogo:

- “Já que você é um professor de Matemática, vou lhe dar uma charada. Hoje meus três filhos celebram seus aniversários e eu gostaria que você adivinhasse suas idades”.
- “Ok”, respondeu o professor. “Mas você precisa me dizer algo sobre eles!”.
- “Bem, a primeira dica é que o produto de suas idades é 36”.

- “Hum. Só isso não dá para resolver. Preciso de mais alguma dica”.
- “A outra dica é que a soma de suas idades é igual ao número de janelas daquele edifício”, responde apontando para um edifício próximo.
- O matemático então responde: “Ainda necessito de mais uma ajuda”.
- “Bem, meu filho mais velho tem olhos azuis”.
- “Já sei quais são as idades”, respondeu o matemático.

Seguindo o mesmo raciocínio do matemático deste exercício, descubra quais são as três idades dos filhos de seu amigo.

1.12.  Determine quais são os possíveis números (no intervalo fechado de 0 a 9) que se substituídos nos símbolos F , I , A , T torna a multiplicação a seguir verdadeira:

$$\begin{array}{r} IF \\ \times AT \\ \hline FIAT \end{array}$$

Os valores de F , I , A , T devem ser diferentes entre si.

LEI DO DIREITO AUTORAL
Todos os direitos reservados e protegidos
pela Lei 9.610/1998.

Este arquivo não pode ser reproduzido ou
transmitido sejam quais forem os meios
empregados: eletrônicos, mecânicos,
fotográficos ou quaisquer outros.