



LÓGICA DE PROGRAMAÇÃO E ALGORITMOS

AULA 1



Prof. Sandro de Araujo



CONVERSA INICIAL

Esta aula tem como base livros como: *Lógica de programação e estruturas de dados*, *Fundamentos da programação de computadores*, *C Como programar*, *Lógica de programação: a construção de algoritmos e estrutura de dados* e *Lógica de programação algorítmica*, todos contidos em nossa Biblioteca Virtual Pearson. Não deixe de revisá-los e/ou consultá-los em caso de dúvidas.

Esta é a estrutura de conteúdo desta aula:

1. Introdução à lógica;
2. Introdução aos algoritmos;
3. Formas de representação de algoritmos;
4. Construção de algoritmos.

O objetivo da aula é introduzir os principais conceitos e temas das abordagens sobre lógica de programação e algoritmos para a resolução de diferentes problemas, bem como especificar sua importância em problemas computacionais; abordar os conceitos de entrada, processamento e saída, sob o ponto de vista computacional; entender os tipos de algoritmos a serem utilizados nesta disciplina.

TEMA 1 – INTRODUÇÃO À LÓGICA

Neste momento, faremos a seguinte reflexão sobre o significado da palavra “lógica”: **o que significa lógica?**

A lógica objetiva a organização do pensamento e é vista como a arte de pensar. Seu estudo baseia-se em argumentos compostos por premissas e conclusões. Se pararmos para pensar, utilizamos essas premissas e conclusões de forma natural todos os dias em nossas rotinas, por exemplo na simples atividade de colocar uma camisa no guarda-roupa.

Nos exemplos a seguir, temos em **p1** e **p2** as premissas e em **r**, a conclusão:

- a. **p1**: Sei que a camisa está na gaveta da cômoda.
p2: Sei que a gaveta está fechada.
r: Logo, concluo que tenho de abrir a gaveta para pegar a camisa.
- b. **p1**: Sei que sou mais velho que Maria.
p2: Sei que Maria é mais velha que João.



r: Então, concluo que eu sou mais velho que João.

Por trás das nossas rotinas, há sempre o chamado raciocínio lógico, que define os passos para que nossa atividade seja concluída com sucesso. Quando esse raciocínio falha, ou seja, a premissa não é suficiente, há grandes chances de não termos uma conclusão de acordo com o planejado. A mesma coisa acontece com os programas de computador: se a premissa não for satisfatória, o objetivo não será alcançado (Puga; Rissetti, 2016).

Embora pareça simples o uso do raciocínio lógico, é uma tarefa bastante complexa e que exige muita prática. Faça uma reflexão de todos os passos envolvidos na troca de um pneu e os analise. Percebeu que essa tarefa não se resume em tirar um pneu e colocar o outro? Se detalharmos um pouquinho os passos para a conclusão da troca do pneu, teremos algo como:

1. O estepe está cheio o suficiente?
2. Se o estepe estiver cheio...
3. Pegar o macaco no porta-malas.
4. Posicionar o macaco no lugar adequado.
5. Suspender o carro.
6. Retirar todos os parafusos.
7. Retirar o pneu.
8. Colocar o estepe.
9. Parafusar todos os parafusos.
10. Abaixar o macaco.
11. Guardar o macaco no porta-malas.
12. Guardar o pneu furado no porta-malas.
13. Fechar o porta-malas.
14. Entrar no carro.
15. Ligar o carro.
16. Dirigir até encontrar uma borracharia.

Temos na troca do pneu um algoritmo no qual cada instrução é um passo para ter sucesso na tarefa. Do mesmo, modo na preparação de um bolo segue-se uma receita com uma sequência de passos:

1. Bata quatro claras em neve.
2. Adicione duas xícaras de açúcar.



3. Adicione duas xícaras de farinha de trigo, quatro gemas, uma colher de fermento e duas colheres de chocolate.
4. Bata por três minutos.
5. Unte uma assadeira com manteiga e farinha de trigo.
6. Coloque a massa na assadeira.
7. Leve ao forno para assar durante 20 minutos em temperatura média.

Agimos quase automaticamente na execução das tarefas no nosso dia a dia e não prestamos atenção em todos os passos envolvidos em uma simples atividade, como no exemplo da troca de pneu de um carro. Porém, cada tarefa a ser executada pelo computador deve ser minuciosamente detalhada pelo programador, que parte do princípio de que o computador não pensa e não é inteligente para saber o que precisa ser feito e como deve ser feito (Forbellone; Eberspacher, 2005; Puga; Rissetti, 2016).

Vejamos outros exemplos (Guedes, 2014):

- **Algoritmo – somar três números.**
 - ✓ Passo 1 – receber os três números.
 - ✓ Passo 2 – somar os três números.
 - ✓ Passo 3 – mostrar o resultado obtido.
- **Algoritmo – fazer um sanduíche.**
 - ✓ Passo 1 – pegar o pão.
 - ✓ Passo 2 – cortar o pão ao meio.
 - ✓ Passo 3 – pegar maionese.
 - ✓ Passo 4 – passar maionese no pão.
 - ✓ Passo 5 – pegar a alface e o tomate.
 - ✓ Passo 6 – cortar a alface e o tomate.
 - ✓ Passo 7 – colocar a alface e o tomate no pão.
 - ✓ Passo 8 – pegar o hambúrguer.
 - ✓ Passo 9 – fritar o hambúrguer.
 - ✓ Passo 10 – colocar o hambúrguer no pão.
- **Algoritmo – ir para a escola.**
 - ✓ Passo 1 – acordar cedo.
 - ✓ Passo 2 – ir ao banheiro.
 - ✓ Passo 3 – abrir o armário para escolher a roupa.



- ✓ Passo 4 – se estiver quente, pegar uma camiseta e uma calça jeans; caso contrário, pegar um agasalho e uma calça jeans.
- ✓ Passo 5 – vestir a roupa escolhida.
- ✓ Passo 6 – tomar café.
- ✓ Passo 7 – pegar uma condução.
- ✓ Passo 8 – descer próximo à escola.
- **Algoritmo – atravessar a rua.**
 - ✓ Passo 1 – olhar para a direita.
 - ✓ Passo 2 – olhar para a esquerda.
 - ✓ Passo 3 – se estiver vindo um carro, espere; caso contrário, atravesse a rua.
- **Algoritmo – sacar dinheiro no caixa eletrônico.**
 - ✓ Passo 1 – ir até o caixa eletrônico.
 - ✓ Passo 2 – colocar o cartão no local indicado.
 - ✓ Passo 3 – digitar a senha.
 - ✓ Passo 4 – solicitar a quantia desejada.
 - ✓ Passo 5 – se o saldo for maior ou igual à quantia desejada, sacar; caso contrário, ler mensagem de impossibilidade de saque.
 - ✓ Passo 6 – retirar o cartão.
 - ✓ Passo 7 – sair do caixa eletrônico.

Do mesmo modo, esses detalhamentos se dão na organização dos passos que são ordenados em uma estrutura de controle para a realização das tarefas no computador. A sequência de passos ordenados para realização de uma tarefa é o que chamamos de algoritmo.

TEMA 2 – INTRODUÇÃO AOS ALGORITMOS

Algoritmo é uma sequência lógica de passos que visam atingir um objetivo bem definido (Forbellone; Eberspacher, 2005). Apesar de a palavra algoritmo parecer estranha e muitas vezes até ser confundida com logaritmo, sabemos o que são e como construir algoritmos. Se não fosse verdade, ninguém sairia de casa pela manhã, você não iria à escola, não decidiria qual é o melhor caminho para chegar a um lugar, entre outros. Para que tais atividades sejam feitas, é necessária uma série de entradas de dados – do tipo a hora para acordar, a hora



para sair de casa, o melhor meio de transporte – para executar os algoritmos (Forbellone; Eberspacher, 2005; Guedes, 2014; Puga; Rissetti, 2016).

Um fator importante é que, às vezes, um problema pode ser resolvido de diversas maneiras, porém gerando a mesma resposta/resultado, ou seja, podem existir vários algoritmos para solucionar o mesmo problema. Por exemplo, para fazer o trajeto de casa até a escola, podemos escolher entre diversos meios de transporte (ônibus, carro, moto, caminhada, bicicleta, entre outros) em função de melhor preço, maior conforto, maior rapidez, entre outros. Esses critérios influenciarão diretamente a escolha de quais passos seguir na tomada de decisão, mas o resultado, que é chegar à escola, é o mesmo para todos os meios de transporte disponíveis. Veja os exemplos na Tabela 1.

Tabela 1 – Algoritmos distintos para a solução de um mesmo problema.

| Ir para a escola | | |
|--|--|-------------------------------------|
| Ônibus (mais barato) | Caminhada (mais saudável) | Carro (mais confortável) |
| INÍCIO | INÍCIO | INÍCIO |
| 1. Caminhar até o ponto de ônibus. | 1. Colocar um calçado adequado para caminhada. | 1. Entrar no carro. |
| 2. Esperar o ônibus. | 2. Mapear os trajetos até a escola. | 2. Mapear os trajetos até a escola. |
| 3. Entrar no ônibus. | 3. Escolher o trajeto. | 3. Escolher o trajeto. |
| 4. Descer no ponto mais próximo da escola. | 4. Caminhar até a escola | 4. Dirigir até a escola. |
| 5. Caminhar até a escola. | 5. Chegar à escola. | 4. Sair do carro. |
| 6. Chegar à escola. | FIM | 6. Chegar à escola. |
| FIM | | FIM |

Temos na Tabela 1 três algoritmos, com critérios diferentes, que obtiveram o mesmo resultado.

TEMA 3 – FORMAS DE REPRESENTAÇÃO DE ALGORITMOS

Para Carvalho (2005), existem várias formas de representar os algoritmos, por exemplo:

- Por meio de representações gráficas como: diagrama de Nassi-Shneiderman, fluxograma, entre outros.
- Por meio de uma descrição narrativa (português, inglês, espanhol etc.): forma utilizada em receitas culinárias, bulas de remédios, manuais de instruções, tutoriais, nos exemplos da Tabela 1, na descrição dos passos dados para a troca de um pneu, entre outros.



- Por meio de um pseudocódigo, uma linguagem de programação (Java, C#, C++, etc.): alguns programadores experientes "pulam" a etapa do algoritmo e vão direto para a linguagem de programação de sua preferência.

Cada uma das três formas de representar um algoritmo acima tem suas vantagens e desvantagens, e a escolha da melhor representação cabe ao programador.

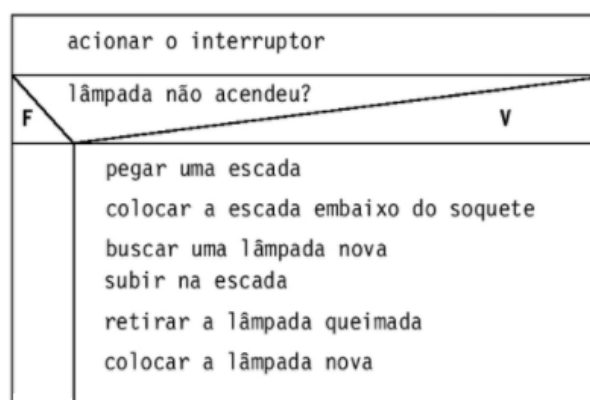
Nos próximos tópicos, apresentam-se as três formas representação mais utilizadas na criação de algoritmos:

- Diagrama de Nassi-Shneiderman.
- Fluxograma.
- Português estruturado ou pseudocódigo.

3.1 Diagrama Nassi-Shneiderman

O diagrama Nassi-Shneiderman, ou diagrama N-S, é uma forma gráfica de representar programas de computador ou algoritmos. Esse diagrama permite a visualização do sistema como um todo e simplifica o desenvolvimento do projeto e a identificação de eventuais falhas no raciocínio. A Figura 1 ilustra o exemplo de um algoritmo de realizar a troca de uma lâmpada.

Figura 1 – Diagrama N-S



Fonte: Adaptado de Forbellone e Eberspacher, 2005.

A ideia básica do diagrama N-S é representar as ações de um algoritmo dentro de um único retângulo, subdividindo-o em retângulos menores, que representam os diferentes blocos de sequência de ações do algoritmo. Entre os blocos, há uma pergunta que influenciará o fluxo das ações, que é "lâmpada não



acendeu?”. Se a **lâmpada acender**, a resposta para a pergunta será **falsa** (representado pela letra “F”) e encerra o fluxo do algoritmo; se a **lâmpada não acender**, a resposta para a pergunta será **verdadeira** (representado pela letra “V”) e segue o fluxo do algoritmo.

- **Vantagens:**

- ✓ Maior clareza no fluxo de execução.
- ✓ Linguagem visualização.

- **Desvantagens:**

- ✓ Requer conhecimento de convenções gráficas.
- ✓ Mais trabalho em decorrência de seus desenhos.
- ✓ Dificuldade para fazer correções.

3.2 Fluxograma

O fluxograma é uma representação gráfica que utiliza formas geométricas ligadas por setas para indica fluxo, ações (instruções) e decisões que deverão ser seguidas para resolver um problema (Guedes, 2014).

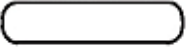

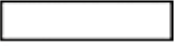
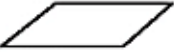


Essas formas geométricas são um tipo de diagrama que pode ser entendido como uma **representação esquemática de um processo**, que ilustra de forma descomplicada a sequência de execução dos elementos que o compõem. Podemos entendê-lo, na prática, como uma documentação dos passos necessários para a execução de um processo qualquer que permite visualizar o fluxo e as etapas de processamento de dados possíveis para a resolução do problema que, segundo os autores Ascencio e De Campos (2012) e Guedes (2014), oferece vantagem e desvantagem na sua adoção:

- **Vantagem** – símbolos gráficos são mais simples de compreender do que textos.
- **Desvantagem** – é preciso aprender a simbologia, e não há detalhes mais precisos, o que dificulta a transcrição para uma linguagem de programação. Além do mais, problemas complexos resultam em um desenho gráfico muito denso, que torna difícil a visualização.

Na Figura 2, apresentam-se os conjuntos de símbolos mais usados para compor um algoritmo.



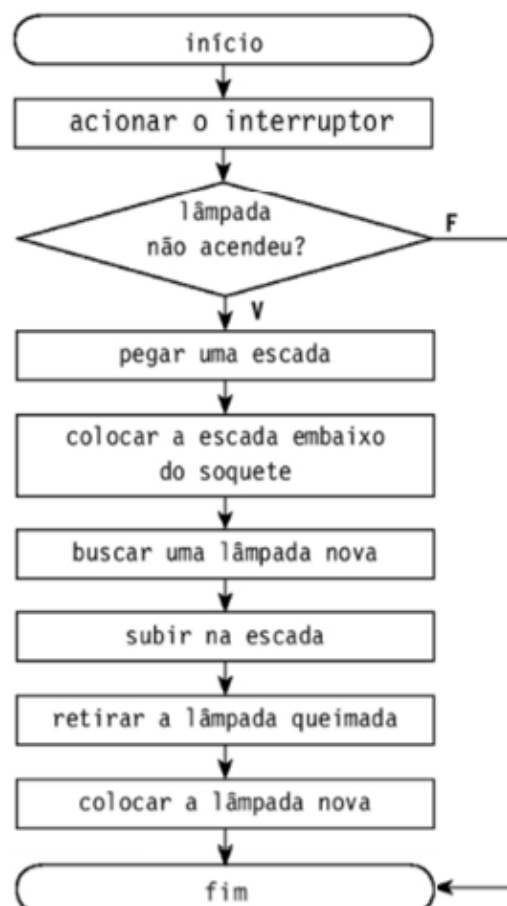
Figura 2 – Conjunto de símbolos utilizados no fluxograma

| | |
|---|---|
|  | Símbolo utilizado para indicar o início e o fim do algoritmo. |
|  | Símbolo que permite indicar o sentido do fluxo de dados. Serve exclusivamente para conectar os símbolos ou blocos existentes. |
|  | Símbolo utilizado para indicar cálculos e atribuições de valores. |
|  | Símbolo utilizado para representar a entrada de dados. |
|  | Símbolo utilizado para representar a saída de dados. |
|  | Símbolo utilizado para indicar que deve ser tomada uma decisão, apontando a possibilidade de desvios. |

Fonte: Ascencio e Campos, 2012.

A seguir, apresenta-se na Figura 3 um fluxograma com os passos necessários para trocar uma lâmpada. Este algoritmo é a mesma sequência de passos usada anteriormente, na Figura 1.

Figura 3 – Fluxograma para trocar uma lâmpada



Fonte: Adaptado de Forbellone e Eberspacher, 2005.



No algoritmo da Figura 3, nota-se um objetivo bem definido, que é **trocar uma lâmpada**. Porém, se a lâmpada não estiver queimada, a execução dos passos para sua troca não será realizada. Nesse algoritmo, a lâmpada só será trocada se realmente estiver queimada.

3.3 Português estruturado

O português estruturado, também conhecido como portugal, pseudocódigo ou pseudolinguagem, é uma forma de linguagem com significados bem definidos de termos utilizados nas instruções do algoritmo, que usa palavras e estruturas com significado predefinido em um padrão a ser seguido. Podemos dizer que é uma linguagem intermediária entre a linguagem natural e uma linguagem de programação usada na construção de programas. A sintaxe do português estruturado possui todos os elementos básicos e a estrutura semelhantes aos de uma linguagem de programação de computadores.

Assim como o fluxograma, o pseudocódigo oferece vantagem e desvantagem na sua adoção (Ascencio; de Campos 2012; Guedes, 2014):

- **Vantagem** – a passagem para qualquer linguagem de programação é quase imediata.
- **Desvantagem** – primeiro é necessário lidar com a lógica de programação e, então, aprender as regras do padrão de pseudocódigo utilizado. Ou seja, na prática, segundo as regras, ao criar um pseudocódigo, você desenvolverá o programa para depois transcrevê-lo para uma linguagem de programação.

Apesar de ser uma desvantagem aprender as regras e transcrever o algoritmo depois, o pseudocódigo é amplamente usado por uma grande parte dos programadores, analistas de sistemas, matemáticos, físicos e outros para descrever seus algoritmos. No caso da computação, é mais comum utilizar um pseudocódigo, por ser mais próximo de uma linguagem do dia a dia, embora bastante simplificada (Guedes, 2014).

No exemplo a seguir é apresentado um algoritmo escrito em pseudocódigo, com as mesmas sequências de passos usadas no diagrama N-S e no fluxograma, das Figuras 1 e 3, respectivamente:



INICIO

1. Acionar o interruptor
2. Se a lâmpada não acender:
 - 2.1. Pegue uma escada;
 - 2.2. Coloque a escada embaixo do soquete;
 - 2.3. Busque uma lâmpada nova;
 - 2.4. Suba na escada;
 - 2.5. Retire a lâmpada queimada;
 - 2.6. Coloque a lâmpada nova.

FIM

Esse algoritmo segue esta estrutura:

1. A demarcação do algoritmo começa a partir da palavra **INICIO**.
2. A demarcação do algoritmo termina com a palavra **FIM**.
3. Entre as palavras **INICIO** e **FIM**, então centrados todos os comandos para a execução do algoritmo, que são levemente deslocados para direita.

Resolver problemas com português estruturado pode ser uma tarefa tão complexa quanto escrever um programa em uma linguagem de programação qualquer, só não tão rígida quanto a sua sintaxe.

Se um destinatário entender os comandos (ações/instruções), o algoritmo será considerado completo; se não estiver claro, precisa ser depurado com novos comandos, que constituirão um aperfeiçoamento do comando inicial.

Caso necessário, devem ser feitos novos depuramentos até que todos os comandos estejam em conformidade e entendíveis pelo destinatário. No algoritmo trocar a lâmpada, poderia ser necessário detalhar algumas ações, como a ação de testar todas as lâmpadas antes de retirar a lâmpada queimada.

Os algoritmos que exemplificaremos a partir de agora são voltados para solucionar problemas de processamento de dados (problemas computáveis, ou seja, que podem ser resolvidos pelo computador). Portanto, apesar de usarmos o português coloquial, a linguagem será mais restrita, uma vez que o destinatário (o computador) entende poucos comandos (ações/instruções).



TEMA 4 – CONSTRUÇÃO DE ALGORITMOS

Podemos pensar em algoritmo como a receita de uma pizza marguerita. Essa sequência finita de instruções em forma de receita dá cabo de uma meta específica que é o preparo de uma deliciosa pizza marguerita.

Para construir um algoritmo, é preciso seguir estes passos:

1. Entender o problema a ser resolvido e destacar os pontos mais importantes e os objetivos que o compõem.
2. Definir os dados de entrada, ou seja, quais dados serão fornecidos e quais objetivos fazem parte do cenário do problema.
3. Definir o processamento, ou seja, quais operações serão executadas e quais são as restrições para essas operações; o processamento deve transformar os dados de entrada em dados de saída e também verificar quais objetos são responsáveis pelas atividades.
4. Definir os dados de saída, ou seja, quais dados são gerados depois do processo.
5. Construir o algoritmo utilizando um dos três tipos apresentados anteriormente (diagrama N-S, fluxograma ou pseudocódigo).
6. Testar o algoritmo realizando simulações.
7. Corrigir possíveis erros e voltar ao item 5.

Com base nos conceitos vistos até aqui, iniciaremos nossos estudos com o algoritmo representado pelo pseudocódigo, por representar um algoritmo de forma semelhante à das linguagens de programação.

A estrutura de um algoritmo em pseudocódigo pode variar um pouco de acordo com o autor ou com base na linguagem de programação que será utilizada posteriormente, mas essas variações ocorrem apenas na sintaxe, pois a semântica deve ser exatamente a mesma.

A estrutura que empregaremos para a construção de nossos pseudocódigos será a seguinte: 1. algoritmo “nome”; 2. var; 3. inicio; 4. finalgoritmo.

algoritmo “nome” /* Tem como objetivo identificar o algoritmo; deve-se utilizar um nome o mais significativo possível para facilitar a identificação. */

var /* Seção de Declarações – neste ponto são informadas quais variáveis, e seus respectivos tipos, serão utilizadas no algoritmo. */



inicio /* Seção de Comandos – aqui será escrita a sequência de comandos que deve ser executada para solucionar o problema em questão. */
fimalgoritmo // Marca o final do algoritmo.

Observe que palavras que fazem parte da sintaxe da linguagem são palavras reservadas, ou seja, não podem ser usadas para outro propósito em um algoritmo que não seja aquele previsto nas regras de sintaxe. As palavras **algoritmo**, **var**, **inicio** e **fimalgoritmo** são exemplos de palavras reservadas.

Vejamos alguns exemplos de algoritmos usados para resolver problemas computacionais:

Exemplo 1:

Observa-se agora um pseudocódigo que recebeu um valor inteiro, fornecido pelo usuário, e o retornou no monitor.

algoritmo “exemplo 1”

var x: inteiro

inicio

 leia (x)

 escreva (x)

fimalgoritmo

Exemplo 2:

Neste exemplo, o algoritmo que recebe um valor inteiro acresce duas unidades a este e exibe o resultado dessa manipulação.

algoritmo “exemplo 2”

var n: inteiro

inicio

 escreva (“Digite um número inteiro: ”)

 leia (n) $n \leftarrow n + 2$

 escreval (“Resultado (número + 2): ”, n)

fimalgoritmo

Exemplo 3:

O próximo exemplo mostra um pseudocódigo para representar um algoritmo que efetua a multiplicação de dois inteiros quaisquer.

algoritmo “exemplo 3”

var n1, n2, res: inteiro



inicio

```
escreva ("Digite o multiplicando inteiro: ")
leia (n1)
escreva ("Digite o multiplicador inteiro: ")
leia (n2) res <- n1 * n2
escreva ("Resultado da multiplicação: ", res)
```

fimalgoritmo

Observe que, em todos os exemplos mostrados, a ordem lógica da execução das tarefas foi importante para que os algoritmos fossem executados sem erros. Essas tarefas são o que chamamos de instruções organizadas dentro de um escopo limitado pelas palavras **início** e **fim**. Um algoritmo tem que ter um detalhamento para que possa prever todos os passos necessários para a resolução de um problema.

FINALIZANDO

Nesta aula, aprendemos o conceito de algoritmo e começamos a desenvolver o entendimento da lógica a partir de exemplos cotidianos que podem acontecer em qualquer casa, como na troca de uma lâmpada que foi mostrada nas representações de algoritmos estudadas nesta aula. Percebemos, também, que existem várias formas de representar um algoritmo. Essas representações trazem passos que vão solucionar o problema. Vimos, ainda, que um algoritmo deve ser construído dentro de estrutura lógica e organizada, descrito em termos de ações não ambíguas e bem definidas.



REFERÊNCIAS

ASCENCIO, A. F. G.; CAMPOS, E. A. V. de. **Fundamentos da programação de computadores**. São Paulo: Pearson, 2012.

DEITEL, H.; DEITEL, P.; DEITEL. **C como programar**. 6. ed. Pearson, 2011.

FORBELLONE, A. L. V.; EBERSPACHER, H. F. **Lógica de programação: a construção de algoritmos e estrutura de dados**. 3. ed. Prentice Hall, 2005.

GUEDES, S. **Lógica de programação algorítmica**. São Paulo: Pearson, 2014.

PUGA, S.; RISSETTI, G. **Lógica de programação e estruturas de dados**. Pearson, 2016.