Construção de Algoritmos e Programação

Algoritmos para quem já sabe programar

Jander Moreira

24/08/2024

Apresentação

> **!** Importante
>
> Este material está **em produção** e, assim, é necessária atenção quanto à precisão do conteúdo e às possíveis alterações que serão promovidas ao longo do tempo.
> Versão 0.1-alfa

A disciplina de graduação *Construção de Algoritmos e Programação*, que é ofertada regularmente pelo Departamento de Computação para os cursos de Bacharelado em Ciência da Computação e Bacharelado em Engenharia da Computação da Universidade Federal de São Carlos, motivou a escrita deste livro, pensando em uma abordagem distinta da usualmente feita em cursos básicos de programação.

A versão *Algoritmos para quem já sabe programar*

Não é incomum, ainda mais em tempos recentes, que as pessoas já tenham, de alguma forma, tido contato com uma linguagem de programação e já tenham produzidos programas. Ambientes de desenvolvimento de código cada vez mais sofisticados, tanto executados localmente quanto disponíveis *online*, ajudam na escrita do código fonte, dando suporte relevante ao programador.

Programas de qualidade, porém, envolvem em um raciocínio para se chegar a uma boa solução. Isso pode ser esquecido quando o código final passa a ser o foco da atenção em detrimento de uma boa organização da solução do problema que será tratado.

Neste texto, um caminho que envolve a abordagem de problemas e a propostas de soluções algorítmicas para resolvê-los é traçado, supondo que o leitor já possua algum conhecimento em uma linguagem de programação imperativa e estruturada. Embora a linguagem C seja empregada nos exemplos, a experiência prévia com outras linguagens não invalida os conceitos apresentados.

Disponibilidade *online*

- Algoritmos para quem já sabe programar: https://jandermoreira.github.io/cap-algoritmos
- Programação em C : https://jandermoreira.github.io/cap-linguagem-c
- Prática com algoritmos: https://jandermoreira.github.io/cap-pratica-algoritmos

Algoritmos para quem já sabe programar

Escrever algoritmos é uma habilidade importante tanto para cientistas da computação quanto outros profissionais que representem soluções para problemas computacionais, como analistas, engenheiros e muitos outros.

A programação se dá em diferentes linguagens, como Python, C, C++, Java, PHP e uma infinidade delas. Cada linguagem de programação possui suas próprias características intrínsecas, como sintaxe, estruturas de dados e recursos em bibliotecas ou módulos externos, por exemplo. Também as linguagens se apresentam com diferentes graus de abstração: enquanto Python é uma linguagem mais genérica (chamada de *nível alto*), C está mais próxima da representação de memória e das características da máquina (com *nível mais baixo*, embora não seja dependente do conjunto de instruções do processador).

Algoritmos, por sua vez, são representações abstratas e independentes de linguagem de programação. Podem ser apresentados em nível muito alto de abstração e podem, conforme a necessidade, aproximar-se da representação mais próxima ao hardware. Embora haja regras na escrita de algoritmos, elas nunca serão tão rígidas quanto à sintaxe imposta por uma linguagem de programação.

Conteúdo

Parte I

Representação e uso de algoritmos

1 Noções de algoritmos

Um algoritmo é uma descrição de passos que, se seguidos, levam à solução de um determinado problema e, genericamente falando, quaisquer instruções, orientações ou coisa similar podem ser classificadas como um algoritmo.

A partir dessa óptica, há uma infinidade de algoritmos. Quando se trata de computação e mais especificamente de programação, há um subconjunto dos algoritmos com características mais particulares. Este capítulo trata da contextualização dos algoritmos computacionais.

1.1 Características gerais dos algoritmos

Por exemplo, as instruções de como higienizar uma máquina de lavar roupas são um algoritmo. Esse algoritmo parte do pressuposto de uma máquina de lavar para ser higienizada, descreve os passos para serem seguidos (deixar o cesto vazio, acrescentar a água sanitária, deixar em um ciclo específico por um determinado tempo) e atinge o resultado desejado, que é a máquina limpa.

A montagem de uma estante comprada *online* também segue o algoritmo estabelecido no manual de montagem, tendo como objetivo partir de um conjunto de peças separadas e obter a estante montada e funcional. Os passos passam pela verificação da disponibilidade de todas as peças e ferramentas necessárias, montagem organizada das diversas partes e devidas finalizações.

Um último (e clássico) exemplo é uma receita culinária, a qual parte dos ingredientes constituintes e chega a um bolo, um assado ou outro prato qualquer.

Todos esses algoritmos possuem três elementos principais:

- A situação inicial;
- A sequência de passos que devem ser seguidos;
- A situação final.

A situação inicial são as pré-condições, ou seja, o que tem haver antes da execução dos passos para que todas as ações possam ser seguidas de forma adequada. Os passos determinam as ações que devem ser executadas e uma ordem coerente para que aconteçam. As pós-condições caracterizam a situação final, ou seja, a completude do que o algoritmo se propôs a resolver.

Na Tabela 1.1 são apresentados esses elementos para dois exemplos específicos, ilustrando-os de forma simplificada.

Tabela 1.1: Exemplos ilustrando os elementos (pré-condições, passos, pós-condições) para dois problemas específicos.

	Cocção de um pão	Atualização de um saldo bancário
Pré-condições	Disponibilidade dos ingredientes e utensílios necessários	O saldo anterior e todas as movimentações no período
Passos	Preparação da massa, descanso, crescimento, forno	Atualização do saldo passando-se por cada movimentação individual

	Cocção de um pão	Atualização de um saldo bancário
Pós-condições	Pão	O saldo atualizado

Algoritmo

Um algoritmo pode ser definido como uma sequência finita de passos que levam de uma situação inicial (pré-condições) a uma situação final (pós-condições) de forma bem definida. A partir desse conceito, é esperado que, a partir do mesmo estado inicial e seguidos os mesmos passos, o estado final seja atingido.

Esta definição não se aplica, em particular, à receita do pão indicada na Tabela 1.1. O resultado tende a variar consideravelmente dependendo de uma variedade de situações não mencionadas, como o tipo e a qualidade da farinha, a temperatura ambiente que influencia no crescimento da massa e o forno usado, que pode aquecer mais ou menos que outro forno, por exemplo. Para se garantir um resultado sempre "igual", todas essas variáveis deveriam entrar nas pré-condições. Felizmente, essas variações são toleradas no resultado final da receita, sendo até esperadas tais diferenças. As pré-condições e pós-condições podem, dependendo do caso, ter graus de especificidade variados.

Essa variação de resultados, porém, não é tolerada na atualização do saldo bancário. Dado o mesmo saldo inicial e as mesmas movimentações, o resultado não pode ser diferente sob nenhuma hipótese. Tem que haver uma previsibilidade do resultado. Neste caso, pré e pós-condições são bastante determinísticas.

Os algoritmos com resultados e passos mais maleáveis, que toleram certas variações no resultado final, enquadram-se como algoritmos gerais. Entre eles estão as receitas, instruções de montagem de móveis, orientações para se chegar a um destino com GPS ou instruções de como inserir um novo contato na agenda do telefone. Em todos eles, até a vivência e experiências pessoais de quem os executa podem ter influência no resultado. Há pessoas com ótima mão para fazer bolos, por exemplo.

Se for pedido a um humano que converta 95 Fahrenheit para graus Celsius, ele pode usar seu *smartphone* para abrir uma ferramenta de busca, digitar "quanto é 95 farenheit em celsius" (sim, com o erro de digitação) e obter a resposta de 35°C. Ele poderia estar sem bateria e optado por usar um computador para fazer a busca; poderia também ter escolhido uma ferramenta de busca no lugar de outra; poderia até ter digitado o texto da consulta de diversas outras maneiras distintas. Esse humano poderia também ter boa memória e se lembrar da fórmula, além de ter facilidade para fazer contas de cabeça e dar o resultado sem nenhum outro recurso a não ser ele mesmo.

1.2 Algoritmos computacionais

Existe uma classe particular de algoritmos para os quais há um maior rigor nos mais diversos aspectos e eles não podem depender da experiência ou interpretação de que os executa. Esses são os algoritmos escritos para serem executados, em última instância, por um sistema computacional (processador e memória eletrônicos) e, para tanto, têm que ser extremamente claros e precisos em cada instrução a ser executada, bem como possuem pré-condições e pós-condições bastantes específicas.

Caso um sistema computacional, ou seja, um programa, deva realizar a mesma tarefa, ele tem que ter bem definidas todas as etapas. As pré-condições, por exemplo, definiriam que o valor deveria ser um número real, os passos indicariam o cálculo da conversão e qual seria a expressão usada e, finalmente, o resultado produzido como pós-condição estaria bem definido.

Figura 1.1: Duas representações distintas para uma mesma tarefa: representação por figuras ilustrativas e textual.

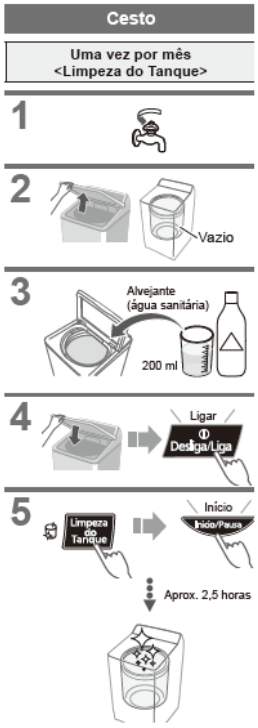
(a) Como limpar máquina de lavar.*

Para limpar a parte interna da máquina, o passo a passo é o seguinte:

- **Passo 1:** primeiramente, selecione o nível alto de água e configure um ciclo longo;
- **Passo 2:** com a máquina cheia, despeje um litro de vinagre de álcool;
- **Passo 3:** deixe a máquina bater durante alguns minutos e pause o ciclo;
- **Passo 4:** adicione uma xícara de chá de bicarbonato de sódio e deixe o composto agir por aproximadamente 30 minutos com a máquina parada;
- **Passo 5**: por fim, inicie o ciclo novamente e deixe ir até o final para o produto ser completamente enxaguado do interior do aparelho.

*Buscapé: https://www.buscape.com.br/lavadora-roupas/conteudo/como-limpar-maquina-de-lavar.

(b) Central de Ajuda Panasonic.*



*Central de Ajuda da Panasonic: https://panasonic-br.zendesk.com/hc/pt-br/articles/360043470732-Como-realizar-a-higieniza%C3%A7%C3%A3o-da-minha-M%C3%A1quina-de-Lavar-roupas-Panasonic-.

Algoritmos computacionais

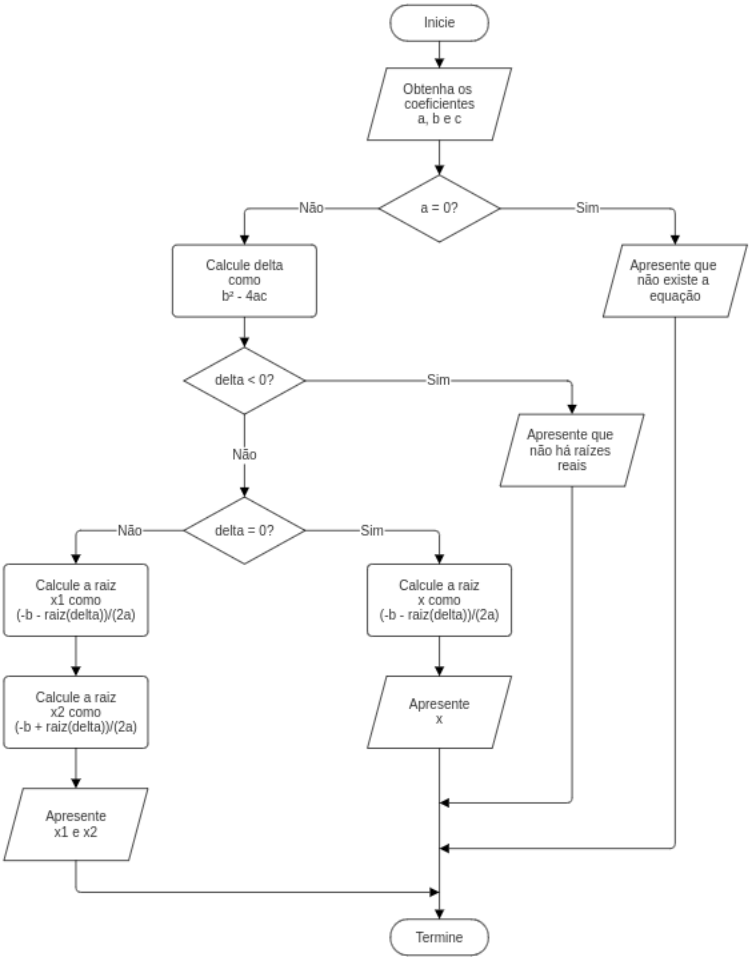
Um algoritmo computacional é aquele que define com clareza todas pré-condições e estabelece também claramente as pós-condições. Também define os passos de forma inequívoca e direta, sem margens para interpretações ou variações. Seu objetivo é ser executado em um sistema computacional.

1.3 As várias formas de representar algoritmos

Existem diversas maneiras de se representar um algoritmo. Para algoritmos gerais, podem ser usadas figuras ou instruções textuais, por exemplo. Na Figura 1.1 são mostradas duas alternativas com algoritmos para a limpeza de uma máquina de lavar roupas. As instruções não são as mesmas, mas o conceito de formas diferentes para que as instruções sejam apresentadas podem variar. Certamente, há um vídeo que também ensina esse procedimento.

Para o caso dos algoritmos computacionais há duas formas mais usuais para representar a sequência de passos que resolvem um dado problema: fluxogramas e pseudocódigo. Naturalmente, há alternativas menos empregadas.

Figura 1.2: Fluxograma para cálculo e apresentação das raízes reais de uma equação de segundo grau.



1.3.1 Fluxogramas

Os fluxogramas são representações visuais com os passos que implementam cada algoritmo. Os símbolos (caixas) possuem formas específicas para cada função e setas as ligam indicando a ordem em que devem ser executadas. Na Figura 1.2 é apresentado um fluxograma para o cálculo das raízes reais de equação de segundo grau e apresentação de mensagens de erro nos casos adequados.

1.3.2 Pseudocódigo

Como alternativa aos fluxogramas, é bastante comum o emprego do chamado pseudocódigo, o qual se assemelha a programas, mas é uma abstração da solução. O Algoritmo 1.1 é apresentado na forma de pseudocódigo.

O Algoritmo 1.1 se refere à mesma solução lógica da Figura 1.2.

1.3.3 Fluxogramas ou pseudocódigo?

Embora não sejam as únicas formas de representar algoritmos, fluxogramas e pseudocódigo são as mais usuais. A questão natural é qual deles escolher.

Os fluxogramas são mais fáceis de serem entendidos, mesmo que por pessoas que não sejam da área de computação, pois apresentam com clareza suficiente fluxo de ações do algoritmo. Quando o

Algoritmo 1.1: Pseudocódigo para o cálculo e apresentação das raízes reais de uma equação de segundo grau.

Descrição: Cálculo e apresentação das raízes reais de uma equação de segundo grau na forma $ax^2 + bx + c = 0$
Requer: Os coeficientes a, b e c da equação
Assegura: as raízes reais da equação; ou mensagem que a equação é inválida; ou mensagem que não há raízes reais

Obtenha os valores de a, b e c ▷ *Coeficientes da equação*
se a for igual a zero **então**
    Apresente que a equação não é do segundo grau
senão
    Calcule o discriminante Δ como $b^2 - 4ac$
    se Δ for negativo **então** ▷ *Não há raízes reais*
        Apresente que não há raízes reais
    senão se Δ for igual a zero **então** ▷ *Apenas uma raiz*
        Calcule x como $-\frac{b}{2a}$
        Apresente o valor de x
    senão ▷ *Duas raízes*
        Calcule x_1 como $\frac{-b - \sqrt{\Delta}}{2a}$
        Calcule x_2 como $\frac{-b + \sqrt{\Delta}}{2a}$
        Apresente x_1 e x_2
    fim se
fim se

nível de complexidade do problema aumenta, porém, os fluxogramas começam a ficar muito densos (muitas caixas e muitas conexões), prejudicando o entendimento. A manutenção dos fluxogramas também é um fator que demanda esforço, mesmo com ferramentas que auxiliem em sua construção. Em geral, fluxogramas possuem nível de abstração elevado, ou seja, simplificam e generalizam conceitos complexos.

Por sua vez, o pseudocódigo é menos abstrato, dado que utiliza uma mistura de linguagem natural com conceitos computacionais para descrever os passos do algoritmo. Em decorrência, faz uso de uma estrutura e uma sintaxe com certa rigidez, uma vez que as instruções para cada ação devem ser exatas, sem ambiguidades ou margem para interpretações diferentes. Os pseudocódigos são descrições estruturadas de algoritmos, possuindo indicações de fluxo bastante rígidas.

Um ponto importante dos pseudocódigos é seu uso frequente em livros e artigos científicos, representando soluções precisas para os problemas estudados.

Este livro faz uso do pseudocódigo para a descrição dos algoritmos, uma vez que a precisão na descrição das soluções é importante.

- ambiguidades
- estruturação
- controle de fluxo

1.4 Algoritmos e programas

O Algoritmo 1.2 é um algoritmo computacional simples com uma solução para a conversão de temperaturas entre duas escalas termométricas: de graus Celsius para Fahrenheit.

Algoritmo 1.2: Conversão de graus Celsius para Fahrenheit.

Descrição: Conversão de escalas termométricas, de graus Celsius para Fahrenheit
Requer: O valor da temperatura em graus Celsius
Assegura: O valor da temperatura em Fahrenheit

Obtenha *celsius*
Calcule *fahrenheit* como $\frac{5}{9}celsius + 32$
Apresente *fahrenheit*

Para exemplificar como esse algoritmo pode se tornar um programa, seguem exemplos de sua implementação em algumas linguagens distintas, sendo importante salientar a grande variação de formatos de comandos e da estrutura de cada linguagem.

Pascal:

```pascal
(*
Conversão de escalas termométricas, de graus Celsius para Fahrenheit
Requer: a temperatura em graus Celsius
Assegura: a temperatura em Fahrenheit
*)
program ConversaoTemperaturas;
var
    Celsius, Fahrenheit: real;
begin
    read(Celsius);
    Fahrenheit := 9 / 5 * Celsius + 32;
    write(Fahrenheit:5:2);
end.
```

Python:

```python
# Conversão de escalas termométricas, de graus Celsius para Fahrenheit
# Pré-condição: a temperatura em graus Celsius
# Pós-condição: a temperatura em Fahrenheit

celsius = float(input())
fahrenheit = 9 / 5 * celsius + 32
print(f"{fahrenheit:.2f}")
```

C:

```c
/*
Conversão de escalas termométricas, de graus Celsius para Fahrenheit
Requer: a temperatura em graus Celsius
Assegura: a temperatura em Fahrenheit
*/
#include <stdio.h>

int main(void) {
    char entrada[160];

    fgets(entrada, sizeof entrada, stdin);
    double celsius;
    sscanf(entrada, "%ld", &celsius);

    double fahrenheit = (double)9 / 5 * celsius + 32;
```

```c
    printf("%.2f", fahrenheit);

    return 0;
}
```

R:

```r
# Conversão de escalas termométricas, de graus Celsius para Fahrenheit
# Pré-condição: a temperatura em graus Celsius
# Pós-condição: a temperatura em Fahrenheit

celsius <- as.numeric(readline(""))
fahrenheit <- celsius * 9 / 5 + 32
cat(fahrenheit, "\n")
```

Java:

```java
// Conversão de escalas termométricas, de graus Celsius para Fahrenheit
// Pré-condição: a temperatura em graus Celsius
// Pós-condição: a temperatura em Fahrenheit

import java.util.Scanner;

public class ConversorTemperatura {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double celsius = scanner.nextDouble();

        double fahrenheit = celsius * 9 / 5 + 32;
        System.out.println(fahrenheit);

        scanner.close();
    }
}
```

Ada:

```ada
-- Conversão de escalas termométricas, de graus Celsius para Fahrenheit
-- Pré-condição: a temperatura em graus Celsius
-- Pós-condição: a temperatura em Fahrenheit

with Ada.Text_IO; use Ada.Text_IO;

procedure Conversor_Temperatura is
   Celsius : Float;
   Fahrenheit : Float;

begin
   Get(Item => Celsius);
   Fahrenheit := Celsius * 9.0 / 5.0 + 32.0;
   Put_Line(Float'Image(Fahrenheit));
end Conversor_Temperatura;
```

Lua:

```lua
-- Conversão de escalas termométricas, de graus Celsius para Fahrenheit
-- Pré-condição: a temperatura em graus Celsius
-- Pós-condição: a temperatura em Fahrenheit

local celsius = tonumber(io.read())
local fahrenheit = celsius * 9 / 5 + 32
print(fahrenheit)
```

Existe, claramente, uma distância entre a solução (algoritmo) e sua implementação (código da linguagem).

O principal conceito por trás dos algoritmos é ter uma solução mais abstrata, a qual não se restringe aos detalhes que cada linguagem impõe e, entretanto, apresenta uma solução simples de entender e objetiva quanto a como o problema abordado é resolvido.

Parte II

Processamento de sequências de dados

2 Estruturas algorítmicas básicas para processamento de sequências de dados

Este capítulo considera o processamento de sequências de dados, o que significa repetir ações sobre cada dado individualmente. Em decorrência, também aborda outras ações que sejam repetitivas.

2.1 Sequências de dados

Uma sequência de dados ocorre quando, por exemplo, quando há uma coleção relativamente grande de dados a serem considerados por um algoritmo.

Para exemplificar, inicialmente é considerado um problema simples: determinar o valor máximo entre três valores reais. Uma solução é dada no Algoritmo 2.1.

Algoritmo 2.1: Determinação do máximo entre três valores reais.

Descrição: Determinação do máximo entre três reais
Requer: três valores reais
Assegura: apresentação do valor máximo

Obtenha os valores de v_1, v_2 e v_3
se $v_1 > v_2$ e $v_1 > v_3$ **então**
    Defina *máximo* como v_1
senão se $v_2 > v_3$ **então**
    Defina *máximo* como v_2
senão
    Defina *máximo* como v_3
fim se
Apresente *máximo*

Há um volume de dados pequeno para esse problema, com apenas três valores. Supondo agora que o problema abordasse não apenas uma trinca de valores, mas uma dezena deles. O Algoritmo 2.2 apresenta uma solução para esse novo problema, seguindo a mesma linha de raciocínio da solução anterior.

Visivelmente é uma solução longa, embora não seja complexa. Entretanto, escrever uma solução similar para 20, 30 ou 100 valores usando essa estratégia é até factível, porém nada prática.

Uma estrutura de organização alternativa para esse problema poderia ser como a apresentada no Algoritmo 2.3.

Para uma quantidade grande de valores, a solução pode ser ampliada apenas pelo acréscimo de novas variáveis v_i e novas verificações. Não é elegante, mas certamente mais prática, e talvez mais clara, que no Algoritmo 2.2.

Um ponto interessante do Algoritmo 2.3 é que, feita a entrada de uma variável e confrontado seu valor com o máximo, ela não é mais usada. Isso leva a uma nova versão: o Algoritmo 2.4, na qual cada

Algoritmo 2.2: Determinação do máximo entre dez valores reais.

Descrição: Determinação do máximo entre três reais
Requer: dez valores reais
Assegura: apresentação do valor máximo

Obtenha os valores de v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9 e v_{10}
se $v_1 > v_2$ e $v_1 > v_3$ e $v_1 > v_4$ e $v_1 > v_5$ e $v_1 > v_6$ e $v_1 > v_7$ e $v_1 > v_8$ e $v_1 > v_9$ e $v_1 > v_{10}$ **então**
    Defina *máximo* como v_1
senão se $v_2 > v_3$ e $v_2 > v_4$ e $v_2 > v_5$ e $v_2 > v_6$ e $v_2 > v_7$ e $v_2 > v_8$ e $v_2 > v_9$ e $v_2 > v_{10}$ **então**
    Defina *máximo* como v_2
senão se $v_3 > v_4$ e $v_3 > v_5$ e $v_3 > v_6$ e $v_3 > v_7$ e $v_3 > v_8$ e $v_3 > v_9$ e $v_3 > v_{10}$ **então**
    Defina *máximo* como v_3
senão se $v_4 > v_5$ e $v_4 > v_6$ e $v_4 > v_7$ e $v_4 > v_8$ e $v_4 > v_9$ e $v_4 > v_{10}$ **então**
    Defina *máximo* como v_4
senão se $v_5 > v_6$ e $v_5 > v_7$ e $v_5 > v_8$ e $v_5 > v_9$ e $v_5 > v_{10}$ **então**
    Defina *máximo* como v_5
senão se $v_6 > v_7$ e $v_6 > v_8$ e $v_6 > v_9$ e $v_6 > v_{10}$ **então**
    Defina *máximo* como v_6
senão se $v_7 > v_8$ e $v_7 > v_9$ e $v_7 > v_{10}$ **então**
    Defina *máximo* como v_7
senão se $v_8 > v_9$ e $v_8 > v_{10}$ **então**
    Defina *máximo* como v_8
senão se $v_9 > v_{10}$ **então**
    Defina *máximo* como v_9
senão
    Defina *máximo* como v_{10}
fim se
Apresente *máximo*

nova obtenção de valor reusa a mesma variável, substituindo seu valor por um novo da sequência de entrada.

Para essa versão, aumentar a quantidade de itens a serem processados pode ser feita apenas acrescentando-se novas leituras e novas comparações, algo como um "copia-e-colar". Entretanto, ainda a solução não é minimamente elegante, mesmo sendo mais prática que as anteriores.

Uma estrutura de repetição é uma ferramenta usada no pseudocódigo para indicar que algumas ações serão executadas várias vezes. Assim, A versão algorítmica final para o problema do máximo entre 10 valores pode ser dada pelo Algoritmo 2.5.

A alteração desse algoritmo para processar, por exemplo, uma sequência de 1000 valores, requer apenas que a linha que indica "do segundo até o décimo" seja reescrita "do segundo até do milésimo", contemplando uma solução para um novo problema, que é bem maior que os anteriores.

2.2 Caracterização das sequências

Nos problemas computacionais, as sequências se apresentam de algumas formas bem características. Conquanto não sendo as únicas apresentações com que sequências podem assumir, estas são as mais comuns:

- as que simplesmente acabam;
- as com indicação de término com um valor especial (sentinela);

Algoritmo 2.3: Determinação do máximo entre dez valores reais (versão 2).

Descrição: Determinação do máximo entre três reais
Requer: dez valores reais
Assegura: apresentação do valor máximo

```
Obtenha o valor de v1
Defina máximo como v1                          ▷ o primeiro é o máximo para iniciar

Obtenha o valor de v2
se v2 > máximo então
    Defina máximo como v2                      ▷ troca máximo somente se for maior
fim se

Obtenha o valor de v3
se v3 > máximo então
    Defina máximo como v3
fim se

Obtenha o valor de v4
se v4 > máximo então
    Defina máximo como v4
fim se

Obtenha o valor de v5
se v5 > máximo então
    Defina máximo como v5
fim se

Obtenha o valor de v6
se v6 > máximo então
    Defina máximo como v6
fim se

Obtenha o valor de v7
se v7 > máximo então
    Defina máximo como v7
fim se

Obtenha o valor de v8
se v8 > máximo então
    Defina máximo como v8
fim se

Obtenha o valor de v9
se v9 > máximo então
    Defina máximo como v9
fim se

Obtenha o valor de v10
se v10 > máximo então
    Defina máximo como v10
fim se

Apresente máximo
```

Algoritmo 2.4: Determinação do máximo entre dez valores reais (versão 3).

Descrição: Determinação do máximo entre três reais
Requer: dez valores reais
Assegura: apresentação do valor máximo

```
Obtenha o valor de v
Defina máximo como v                    ▷ o primeiro é o máximo para iniciar

Obtenha o valor de v
se v > máximo então
    Defina máximo como v                ▷ troca máximo somente se for maior
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Obtenha o valor de v
se v > máximo então
    Defina máximo como v
fim se

Apresente máximo
```

Algoritmo 2.5: Determinação do máximo entre dez valores reais (versão 4).

Descrição: Determinação do máximo entre três reais
Requer: dez valores reais
Assegura: apresentação do valor máximo

```
Obtenha o valor de v
Defina máximo como v                          ▷ o primeiro é o máximo para iniciar

para cada item da sequência, do segundo até a décimo, faça
    Obtenha o valor do item em v
    se v > máximo então
        Defina máximo como v                  ▷ troca máximo somente se for maior
    fim se
fim para

Apresente máximo
```

- as de comprimento previamente conhecido;
- as prefixadas com a quantidade de itens.

Supondo que haja dados armazenados em um arquivo, a serem processados um a um, eles se caracterizam como uma sequência que, processado o último dado, a repetição sobre eles se encerra. Assim, os dados são processados até que não haja mais nada a processar. Naturalmente, essa perspectiva vale não somente para arquivos, mas também para dados digitados diretamente em um terminal.

Uma alternativa para a organização dos dados é ter um dado especial que indique que a sequência terminou. Isso ocorre, por exemplo, na representação de cadeias de caracteres na linguagem C, sendo usado um caractere nulo (`\0`) para indicar que os bytes da cadeia terminaram. Essa alternativa pode ser empregada para entrada de dados via terminal, quando valores são digitados e um valor especial (como zero, por exemplo), significa que a entrada terminou. Esta forma de representação de sequências requer a escolha de um valor especial, conhecido geralmente como sentinela, o qual não pode fazer parte da sequência.

Algumas sequências possuem comprimento conhecido. Se dados são colhidos diariamente, haverá uma sequência de 365 (ou 366) dados em um ano. Para o processamento, sabe-se previamente quantos dados existem e controlar uma repetição para processá-los pode ser feita mais facilmente.

Sendo uma forma bastante comum, uma sequência pode ser precedida da quantidade de itens que ela contém. Na linguagem Pascal, uma cadeia de caracteres é precedida por um byte que indica seu comprimento. No processamento deste tipo de sequência, o controle da repetição para o processamento é conhecido logo no início. A título de exemplo, este seria o caso de um polígono qualquer, descrito pelas coordenadas de seus vértices, ser precedido da quantidade de vértices, com triângulos precedidos por um 3 e pentágonos, por um 5.

As duas primeiras caracterizações de sequências, as que acabam e as que possuem sentinela, são chamadas repetições indefinidas ou abertas. Nelas, não é conhecido previamente quantas vezes as ações de processamento serão repetidas. Em contraposição, quando o comprimento da sequência é fixo ou precedido de um valor indicando quantos itens são, a repetição se caracteriza como definida, sabendo-se quantas vezes as ações serão executadas.

Dado o universo de dados existentes, estas possibilidades não contemplam, é claro, todas as sequências, mas restringem nichos de grande relevância neste contexto.

2.3 Fluxo de execução com repetições

Nos algoritmos escritos com pseudocódigo há três estruturas clássicas para indicar a repetição, embora não sejam as únicas. Elas são representadas por **enquanto**, **repita** e **para**.

Quando as repetições são indefinidas quanto ao número de vezes, as estruturas **enquanto** e **repita** são empregadas. Repetições sobre sequências de dados para as quais se sabe antecipadamente o número de iterações, ou seja, para as repetições definidas, a estrutura **para** é utilizada.

2.3.1 Fluxo com enquanto

O **enquanto** é uma estrutura que indica uma repetição. Ela se apresenta como apresentado na sequência.

enquanto *condição* **faça**
    ▷ *Instruções condicionadas*
fim enquanto

A lógica que define o fluxo de execução com a estrutura **enquanto** é baseada em uma *condição*, a qual pode ser avaliada como verdadeira ou falsa. Se a condição for verdadeira, todas as *instruções condicionadas* da estrutura são executadas uma vez. Ao final da execução, a condição é avaliada novamente e nova execução das ações internas pode ou não ser feita, a depender do resultado.

Como consequência, a estrutura indica que as *instruções condicionadas* podem ser executadas uma quantidade indefinida de vezes, que pode ser zero ou mais. É, assim, importante destacar que, sendo a *condição* falsa já na primeira verificação, nenhuma instrução interna é executada.

Para que a estrutura tenha sentido prático, é esperado que as instruções alterem de alguma forma dos dados para que, eventualmente, a condição se torne falsa e a repetição se encerre.

O processamento de sequências de dados cuja quantidade de itens não seja previamente conhecida é bastante comum e, assim, uma repetição de obtenção de valores com **enquanto** é bastante apropriada. O Algoritmo 2.6 é um exemplo dessa necessidade. Ele apresenta uma solução para, dadas várias idades de pessoas (podendo não haver nenhuma), determinar quantas delas são maiores ou iguais a 18.

Algoritmo 2.6: Determinação da quantidade de pessoas com maioridade legal.

Descrição: Determinação da quantidade de pessoas com maioridade legal
Requer: uma sequência de zero ou mais idades
Assegura: a quantidade de idades maiores ou iguais a 18

Inicie um contador com valor igual a zero
enquanto há idades na sequência de entrada **faça**
    Obtenha uma *idade*
    se *idade* ≥ 18 **então**
        Acrescente mais um ao contador ▷ *conta maiores de idade*
    fim se
fim enquanto
Apresenta quantidade de idades contadas

O **enquanto** verifica se há dados disponíveis como entrada e, havendo, obtém o dado (uma idade) e, sendo maior ou igual a 18 anos, conta essa ocorrência. Quando não houver mais dados na entrada, a repetição se encerra e a contagem é apresentada. Para o caso da entrada ser vazia, a condição do **enquanto** já é falsa logo no início e, assim, nenhuma leitura é feita, situação em que o contador permanece com valor zero.

O uso do sentinela de dados também é viável de ter sua lógica organizada como **enquanto**. Um exemplo simples é a soma de uma série de números reais, todos maiores que zero. O sentinela será o número zero, que não faz parte dos dados válidos. Uma solução para esse problema é apresentada no Algoritmo 2.7.

Algoritmo 2.7: Soma de uma sequência de números reais com sentinela.

Descrição: Soma de uma sequência de valores reais
Requer: uma sequência de zero ou mais valores maiores que zero seguida por um valor nulo (sentinela)
Assegura: apresentação da soma dos valores ou zero se a sequência for vazia

```
Defina sentinela com valor zero
Obtenha um valor                          ▷ pode ser o primeiro ou o sentinela
Inicie uma soma com zero
enquanto valor ≠ sentinela faça
    Acrescente valor à soma
    Obtenha um valor                      ▷ próximo valor
fim enquanto
Apresenta o valor da soma
```

A estratégia é fazer, antes da repetição, uma primeira leitura, que pode ser um valor válido ou o próprio sentinela (quando sequência de números está vazia). Antes de somar esse valor, é feita a comparação com o sentinela. Sendo o valor zero, nenhuma repetição é efetuada e a soma zero é o resultado apresentado. Caso seja um valor válido, ele é somado e, antes de nova verificação, o próximo valor da sequência de entrada é obtido, fechando um ciclo.

2.3.2 Fluxo com repita

A estrutura **repita** também se aplica a repetições indefinidas. Sua estrutura é como se apresenta a seguir.

```
repita
    ▷ Instruções condicionadas
até que condição
```

O **repita** é uma estrutura que possui uma *condição* a ser verifica e sua avaliação é feita depois da execução dos *comandos condicionados*. Dessa forma, todos os comandos internos são executados e, então, a *condição* é verificada; resultando em falso, uma nova repetição é feita, sendo verdadeira, repetição se encerra.

Por sua estrutura, os comandos internos do **repita** são executados pelo menos uma vez.

Para exemplificar, pode ser considerado o problema da verificação da senha digitada por um usuário para acesso ao sistema, cujas regras são: se a senha digitada for correta, o acesso é dado; caso haja dez erros consecutivos, uma penalização de tempo ("aguarde cinco minutos para tentar novamente") é aplicada. A quantidade de senhas que serão verificadas é desconhecido: se houver a inserção correta da senha logo no início há apenas uma verificação; se houver senhas inválidas, outras verificações serão necessárias.

O Algoritmo 2.8 ilustra como essa questão pode ser abordada.

Nessa solução, a repetição somente se encerra com a digitação da senha correta, o que pode levar uma eternidade. O algoritmo não detalha como a senha é efetivamente verificada, o que não é, porém, relevante no momento.

Algoritmo 2.8: Verificação de senha para acesso a um sistema.

Descrição: Verificação da senha para garantir acesso a um sistema
Requer: uma sequência indefinida de tentativas de senha
Assegura: acesso ao sistema apenas quando a senha estiver correta

```
Defina número_de_erros com valor zero
repita
    Obtenha senha                                              ▷ tentativa de acesso
    se senha está incorreta então
        Some 1 a número_de_erros
        se número_de_erros for igual a 10 então                ▷ 10 erros?
            Apresente mensagem sobre o número de tentativas erradas
            Aguarde cinco minutos de penalidade antes de prosseguir   ▷ penalidade
            Zere número_de_erros                               ▷ reinicia a contagem
        fim se
    fim se
até que senha esteja correta
Libere o acesso ao sistema
```

Sequências de dados com valor sentinela também podem ser processadas usando-se a estrutura **repita**. O problema resolvido pelo Algoritmo 2.7, que usa -enquanto, tem sua versão com **repita** apresentada no Algoritmo 2.9.

Algoritmo 2.9: Soma de uma sequência de números reais com sentinela (versão 2).

Descrição: Soma de uma sequência de valores reais
Requer: uma sequência de zero ou mais valores maiores que zero seguida por um valor nulo (sentinela)
Assegura: apresentação da soma dos valores ou zero se a sequência for vazia

```
Defina sentinela com valor zero
Inicie uma soma com zero
repita
    Obtenha um valor
    se valor ≠ sentinela então                                 ▷ não soma o sentinela
        Acrescente valor à soma
    fim se
até que valor = sentinela                                      ▷ encerra com o sentinela
Apresenta o valor da soma
```

Nessa solução, a cada repetição é obtido um valor da entrada e a soma é feita apenas para os valores que não sejam iguais ao sentinela. Quando o valor sentinela é obtido da entrada, ele é ignorado e, em seguida, a repetição se encerra e o resultado final é apresentado. Caso não haja valores para serem somados, o resultado zero é apresentado.

2.3.3 Fluxo com para

O **para** é indicado para repetições definidas, ou seja, para aquelas em que a quantidade de repetições é conhecida de antemão. A seguir é apresenta a estrutura do **para**.

```
para especificação_da_repetição faça
    ▷ Instruções condicionadas
fim para
```

Há muitas variações comuns para a repetição com **para**. Elas são apresentadas a seguir.

A versão mais comum para um **para** é com o uso de um iterador (normalmente uma variável genérica i estabelecendo o valor inicial e o final. Seguem exemplos.

▷ *Repetição de 10 vezes*
para $i \leftarrow 1$ **até** 10 **faça**
    Execute uma ação
fim para

▷ *Repetição de 10 vezes*
para $i \leftarrow 0$ **até** 9 **faça**
    Execute uma ação
fim para

▷ *Repetição de n vezes*
Obtenha o valor de n
para $i \leftarrow 0$ **até** $n - 1$ **faça**
    Execute uma ação
fim para

O uso do **for** é considerado sempre crescente[1]. Assim, uma repetição de 1 até 1 é feita apenas uma vez e para de 0 até -10 não há nenhuma repetição. Neste livro, portanto, essa é a interpretação consistentemente seguida.

Como exemplo, a repetição seguinte pode ser considerada.

Obtenha o valor de *inicial* e *final*
para $i \leftarrow$ *inicial* **até** *final* **faça**
    Execute uma ação
fim para

Nela, se *final* > *inicial*, a repetição vai de *inicial* até *final* de 1 em 1, em um total de *final* − *inicial* + 1 vezes. Caso *inicial* = *final* as instruções internas são executadas uma única vez. Finalmente, se *inicial* > *final*, não há nenhuma execução.

Repetições regressivas

As repetições com **para** são, em princípio, crescentes. Para especificar uma repetição decrescente é necessário que ela se torne evidente e não requeira interpretações que podem variar entre diferentes pessoas.

Neste livro há duas formas de indicar "contagens regressivas": explicitamente na instrução ou alterando o passo com que a variável de controle da repetição é atualizado.

▷ *Repetição 10 vezes, de 10 até 1*
para $i \leftarrow 10$ **regressivo até** 1 **faça**
    Execute uma ação
fim para

▷ *Repetição 10 vezes, de 10 até 1*
para $i \leftarrow 10$ **até** 1 **passo** -1 **faça**
    Execute uma ação
fim para

[1] Na realidade, nem sempre, mas é a forma de interpretação mais segura, a não ser que haja uma ressalva junto com o algoritmo.

Repetições com passo diferente de 1

Conforme apresentado nos demais exemplos deste capítulo, assume-se por padrão que os incrementos (ou decrementos com **regressivo até**) seja sempre unitários. Ainda assim é possível indicar iterações com passos diferentes.

Seguem alguns exemplos.

▷ *Repetição para 0, 2, 4, 6, 8 e 10*
para $i \leftarrow 0$ **até** 10 **passo** 2 **faça**
    Execute uma ação
fim para

▷ *Repetição para 10, 8, 6, 4, 2 e 0*
para $i \leftarrow 10$ **até** 0 **passo** -2 **faça**
    Execute uma ação
fim para

Embora incomuns[2], repetições com **para** que usem valores reais também podem ser usadas

▷ *Repetição para 0, 0.25, 0.5, 0.75, 1*
para $i \leftarrow 0{,}0$ **até** 1,0 **passo** 0,25 **faça**
    Execute uma ação
fim para

[2]Há uma preferência geral de que repetições que não usem iteradores inteiros sejam escritas com **enquanto** ou **repita**.

3 Desenvolvimento de algoritmos para processamento de sequências de dados

Este capítulo estende a discussão do Capítulo 2 quanto ao uso das estruturas de controle execução repetitiva, apresentando problemas práticos e indicando ações de uso frequente, como contagens e somas, por exemplo.

O contexto principal abordado é o do processamento de sequências de dados, como as caracterizadas na seção Seção 2.2, abordando coleções de dados disponíveis como entrada para algoritmo.

3.1 Ações simples: contagens e somas

O levantamento de informações a partir de coleções de dados é sempre relevante para análises em nível mais alto. Por exemplo, a determinação do número de estudantes que obtiveram nota zero em um exame do ENEM é um indicador relevante, assim como saber a média das provas de matemática por estado da federação. Da forma similar, a identificação de transações acima de um limite em operações bancárias pode ser um indicador relevante para o mercado e para o governo.

A obtenção dessas informações recai, em última (e simplificada) instância, em ações básicas como contar e somar.

3.1.1 Contagem

A contagem nada mais é que a identificação do número de ocorrências. Para exemplificar, um problema bastante trivial é apresentado: determinar, para uma sequência de idades, a quantidade de crianças, o seja, com até 12 anos incompletos[1]. A quantidade de idades disponíveis é considerada desconhecida, podendo também ser nula. O Algoritmo 3.1 apresenta uma solução para processar essa coleção de dados como uma sequência.

Algoritmo 3.1: Determinação do número de crianças em uma coleção de idades zero ou mais itens (versão com imprecisão).

Descrição: Determinação do número de crianças (até 12 anos) a partir de uma sequência de idades
Requer: uma sequência de zero ou mais idades
Assegura: apresentação da quantidade de idades menores de 12 anos nessa sequência

enquanto existem idades para serem processadas **faça**
    Obtenha *idade*
    se *idade* < 12 **então**
        Conte essa idade como uma criança
    fim se
fim enquanto
Apresente a contagem de crianças

[1] Faixa etária definida pelo ESTATUTO... (1990).

O uso da estrutura **enquanto** é relevante, pois controla uma quantidade indeterminada de idades como entrada e trata, também, o caso dessa quantidade ser nula.

Esse algoritmo contém um elemento de imprecisão. No caso, por exemplo, de não haver nenhuma idade abaixo dos 12 anos, a condição do **se** nunca será verdadeira e nenhuma contagem é feita. Porém, a última instrução explicita que que uma contagem que nunca foi feita deva ser apresentada.

É interessante que, nos algoritmos, essa ambiguidade seja explicitamente resolvida e, para isso, a forma com que a contagem é feita pode ser detalhada, indicando claramente o resultado esperado. O resultado da modificação é o Algoritmo 3.2

Algoritmo 3.2: Determinação do número de crianças em uma coleção de idades zero ou mais itens (versão com imprecisão).

Descrição: Determinação do número de crianças (até 12 anos) a partir de uma sequência de idades
Requer: uma sequência de zero ou mais idades
Assegura: apresentação da quantidade de idades menores de 12 anos nessa sequência

```
Inicie contador com zero                          ▷ contador para número de crianças
enquanto existem idades para serem processadas faça
    Obtenha idade
    se idade < 12 então
        Adicione 1 a contador
    fim se
fim enquanto
Apresente o valor de contador
```

Nesta versão, *contador* é uma variável usada para fazer a contagem, começando explicitamente com zero, o que deixa claro o valor que será apresentado caso não haja nenhuma repetição.

O princípio da contagem é a atualização do valor de uma variável com seu sucessor, de forma que "adicione 1 a *contador*" significa $contador \leftarrow contador + 1$.

Contadores podem ser aplicados em diferentes cenários de entrada, como sequências com sentinela ou com comprimento conhecido ou previamente informado.

O mesmo problema base (contagem de idades inferiores a 12) pode ter especificação e solução novas se a sequência possuir um valor especial de término (sentinela). O problema passa a ser determinar o número de crianças em uma sequência de idades que possui um valor sentinela igual a -1.

Para esse novo problema, uma solução é apresentada no Algoritmo 3.3.

Essa solução usa o **enquanto** para controlar a aparição do valor sentinela. Há uma primeira leitura antes da repetição e, caso já apareça o valor sentinela, não é feita nenhuma repetição e o valor da contagem é zero. Se o primeiro valor for uma idade, ela é verificada para a contagem. Antes de nova verificação no **enquanto**, o próximo valor da sequência é obtido e, logo em seguida, verificado na condição do **enquanto**.

O Algoritmo 3.4 é outra solução para a sequência com sentinela usando **repita**.

Com o uso da estrutura **repita**, obrigatoriamente os comandos internos serão executados pelo menos uma vez. Caso a sequência de idades esteja vazia, o valor obtido será o sentinela, o qual é desconsiderado no **se** e a repetição já se encerra no **até que**, o que resulta em *contador* igual a zero. Caso o valor não seja o sentinela, a idade é analisada e o contador incrementado se necessário, implicando em novo ciclo de repetição.

A questão de contagem de idades pode, por exemplo, também ser aplicada a um problema para o qual a quantidade de idades a ser analisada seja predefinido. Supondo agora a contagem de crianças

Algoritmo 3.3: Determinação do número de crianças em uma coleção de idades zero ou mais itens com sentinela (versão 1).

Descrição: Determinação do número de crianças (até 12 anos) a partir de uma sequência de idades
Requer: uma sequência de zero ou mais idades seguida por um valor sentinela (-1)
Assegura: apresentação da quantidade de idades menores de 12 anos nessa sequência

```
Defina sentinela como valor −1
Inicie contador com zero
Obtenha idade                                   ▷ primeira idade ou sentinela
enquanto idade é diferente de sentinela faça
    se idade < 12 então
        Adicione 1 a contador
    fim se

    Obtenha idade                               ▷ próxima idade ou sentinela
fim enquanto
Apresente o valor de contador
```

Algoritmo 3.4: Determinação do número de crianças em uma coleção de idades zero ou mais itens com sentinela (versão 2).

Descrição: Determinação do número de crianças (até 12 anos) a partir de uma sequência de idades
Requer: uma sequência de zero ou mais idades seguida por um valor sentinela (-1)
Assegura: apresentação da quantidade de idades menores de 12 anos nessa sequência

```
Defina sentinela como valor −1
Inicie contador com zero
repita
    Obtenha idade                               ▷ idade ou sentinela
    se idade não é sentinela e idade < 12 então
        Adicione 1 a contador
    fim se
até que idade seja igual a sentinela
Apresente o valor de contador
```

para uma coleção com quantidade fixa de 100 idades, a solução pode ser dada pelo Algoritmo 3.5. Como o valor é fixo, a estrutura **para** é empregada para a repetição.

Como outra possibilidade, o problema pode considerar uma sequência para a qual seja informada sua quantidade. Por exemplo, para uma sequência de 10 idades, a quantidade 10 é conhecida antes da repetição. O@alg-quantidade-criancas-sequencia-informada considera que o comprimento da sequência esteja disponível antes de cada idade.

Nesse algoritmo, os requisitos são a quantidade e os valores das idades. Antes da repetição, o número de itens deve ser obtido (variável *quantidade*) e a repetição, usando um **para**, executa a obtenção e a análise de cada idade na quantidade de vezes informada.

Nas duas soluções que usam o **para**, a variável de controle da repetição (i) é irrelevante para a solução do problema, sendo usada exclusivamente como indicador para o número de execuções.

Algoritmo 3.5: Determinação do número de crianças em uma coleção de 100 idades.

Descrição: Determinação do número de crianças (até 12 anos) a partir de uma sequência de 100 idades
Requer: uma sequência de 100 idades
Assegura: apresentação da quantidade de idades menores de 12 anos nessa sequência

Inicie *contador* com zero
para $i \leftarrow 1$ **até** 100 **faça**
    Obtenha *idade*
    se *idade* < 12 **então**
        Adicione 1 a *contador*
    fim se
fim para
Apresente o valor de *contador*

Algoritmo 3.6: Determinação do número de crianças em uma coleção de idades com zero ou mais itens precedida pela quantidade de itens.

Descrição: Determinação do número de crianças (até 12 anos) a partir de uma sequência de idades precedida por sua quantidade de itens
Requer: a quantidade de idades e a sequência de idades com a quantidade indicada
Assegura: apresentação da quantidade de idades menores de 12 anos nessa sequência

Inicie *contador* com zero
Obtenha *quantidade*
para $i \leftarrow 1$ **até** *quantidade* **faça**
    Obtenha *idade*
    se *idade* < 12 **então**
        Adicione 1 a *contador*
    fim se
fim para
Apresente o valor de *contador*

3.1.2 Somas

No processamento de volumes de dados, os totais também são uma informação de interesse comumente usada. Por exemplo, nas transações de venda de um estabelecimento, saber o valor total de vendas em um dia pode ser relevante. Também são importantes os volumes de chuva ao longo de um dado período, obtido pela soma das precipitações diárias. Assim, somas são o tema desta seção.

Para explorar somas em sequências, o princípio é similares ao das contagens da Seção 3.1.1, com o reuso de uma variável com a função de acumular valores. Por exemplo, as instruções seguintes ilustram uma forma de reuso para uma soma.

$soma \leftarrow 12$ ▷ *valor inicial*
$soma \leftarrow soma + 6$ ▷ *armazena* $12 + 6 = 18$
$soma \leftarrow soma + 13$ ▷ *armazena* $18 + 13 = 31$

Esta sequência de ações gradativamente calculam $12 + 18 + 31$.

O problema exemplo a ser resolvido agora envolve a apresentação das entradas e saídas de um caixa ao longo do dia. Cada transação feita é registrada por um valor em reais, sendo valores positivos indicativos de entrada de dinheiro no caixa e, em consequência, valores negativos, as saídas. Como

transações com valor nulo não fazem sentido neste contexto, o valor R$0,00 é usado como sentinela para indicar o fim da sequência de valores de transações. O objetivo é apresentar o total de entradas e de saídas, além da diferença entre eles.

Uma solução para esse problema está descrita no Algoritmo 3.7.

Algoritmo 3.7: Cálculo do montante de entrada e de saída a partir de uma sequência de valores de transações (com sentinela).

Descrição: Cálculo, a partir de uma sequência de transações (valores positivos como entrada, negativos como saída), do montante total de entradas e de saída, mais a diferença entre eles
Requer: uma sequência possivelmente vazia de valores de transações seguida por um sentinela (valor 0,00)
Assegura: apresentação do total de entradas, do total de saídas e da diferença entre eles

```
Defina sentinela com o valor zero
Inicie os acumuladores soma_entradas e soma_saídas com zero

Obtenha valor_transação
enquanto valor_transação ≠ sentinela faça
    se valor_transação for positivo então
        Atualize soma_entradas com soma_entradas + valor_transação
    senão
        Atualize soma_saídas com soma_saídas + |valor_transação|      ▷ acumula valor absoluto
    fim se

    Obtenha valor_transação                                          ▷ próxima
fim enquanto

Apresente soma_entradas e soma_saídas                                ▷ totais
Apresente soma_entradas − soma_saídas                                ▷ diferença
```

A estratégia de varredura dos valores das transações é o mesmo do Algoritmo 3.3, com a obtenção do primeiro valor antes do **enquanto** e a obtenção de cada sucessor no final, antes de nova verificação da condição.

A iniciação dos somadores *soma_entradas* e *soma_saídas* é feita com zero. A cada transação, um ou outro tem seu valor aumentado pelo montante da transação atual.

3.1.3 Mínimos e máximos

Valores extremos são itens usualmente buscados em coleções de dados, como, por exemplo, a transação de maior valor no mercado imobiliário ou a idade mínima para um conjunto de pessoas.

A estratégia por trás da determinação dos valores mínimo e máximo é simples. Na procura pelo valor mínimo, cada novo valor é comparado ao mínimo atual e, sendo menor, o mínimo atual é atualizado. Situação análoga ocorre com o valor máximo, substituído apenas quando encontrar um valor maior. Algoritmicamente, essa verificação para o valor mínimo pode ser escrita como indicado na sequência.

```
se novo_valor < valor_mínimo então
    valor_mínimo ← novo_valor
fim se
```

Uma questão que surge é sobre qual deve ser o valor inicial da variável quem mantém o mínimo ou o máximo. A resposta, naturalmente, é que a variável deve ser iniciada com um valor que garanta que as substituições ocorram corretamente.

Uma estratégia segura é sempre adotar o primeiro valor como o valor extremo, seja ele o mínimo ou o máximo, e depois verificar os demais itens.

O problema para exemplificar a localização dos extremos é a determinação, para um coleção de idades, qual o mínimo e o máximo. As idades estão disponíveis em uma sequência precedida por sua quantidade e há pelo menos um valor na sequência.

Algoritmo 3.8: Determinação da idade mínima e máxima em uma coleção de idades precedida pelo número de itens.

Descrição: Determinação da idade mínima e máxima em uma coleção de idades
Requer: a quantidade de idades (maior que zero) seguida dos valores das idades
Assegura: apresentação das idades mínima e máxima

Obtenha *quantidade*
Obtenha *idade* ▷ *primeira idade*
Inicie $idade_{min}$ com *idade* ▷ *mínima até o momento*
Inicie $idade_{max}$ com *idade* ▷ *máxima até o momento*

para $i \leftarrow 2$ **até** *quantidade* **faça** ▷ *para as idades restantes*
    Obtenha *idade*
    se $idade < idade_{min}$ **então**
        Atualize $idade_{min}$ com o valor de *idade* ▷ *atualiza*
    fim se
    se $idade > idade_{max}$ **então**
        Atualize $idade_{max}$ com o valor de *idade* ▷ *atualiza*
    fim se
fim para

Apresente $idade_{min}$ e $idade_{max}$

Uma alternativa comum é iniciar com valores que certamente serão substituídos, como apresentado no Algoritmo 3.9.

Nesta solução, qualquer que seja a primeira idade, ela certamente será menor que $+\infty$ e maior que $-\infty$, forçando as substituições de $idade_{min}$ e $idade_{max}$. No caso de idades, as iniciações poderiam ser, respectivamente, com 200 anos (com uma boa folga) e -1, ambos valores fora do intervalo de idades aceitável.

3.1.4 Derivações

Em decorrência de contagens e somatórios, outros cálculos seguem diretamente, com médias e porcentagens.

Médias

O cálculo de uma média consiste na soma dos valores seguida da divisão pela quantidade. Na prática, é um somatório e uma contagem seguida do cálculo da razão entre eles.

Algoritmo 3.9: Determinação da idade mínima e máxima em uma coleção de idades precedida pelo número de itens.

Descrição: Determinação da idade mínima e máxima em uma coleção de idades
Requer: a quantidade de idades (maior que zero) seguida dos valores das idades
Assegura: apresentação das idades mínima e máxima

Obtenha *quantidade*
Inicie $idade_{min}$ com $+\infty$
Inicie $idade_{max}$ com $-\infty$
para $i \leftarrow 1$ **até** *quantidade* **faça** ▷ *para todas as idades*
    Obtenha *idade*
    se $idade < idade_{min}$ **então**
        Atualize $idade_{min}$ com o valor de *idade* ▷ *atualiza*
    fim se
    se $idade > idade_{max}$ **então**
        Atualize $idade_{max}$ com o valor de *idade* ▷ *atualiza*
    fim se
fim para

Apresente $idade_{min}$ e $idade_{max}$

Supondo que o problema seja o processamento de uma sequência simples de pontuações, todas de 0 a 100, e se deseja a pontuação média. O Algoritmo 3.10 apresenta uma solução para o problema, considerando que a sequência de entrada contém pelo menos uma pontuação.

Algoritmo 3.10: Determinação da pontuação média para uma coleção de pontuações.

Descrição: Determinação da média de uma sequência de pontuações
Requer: a sequência não vazia das pontuações
Assegura: apresentação da pontuação média

Inicie *contador* e *soma* com zero
enquanto há pontuações a serem processadas **faça**
    Obtenha *pontuação*
    Adicione *pontuação* a *soma* ▷ *acumula*
    Adicione 1 a *contador* ▷ *conta*
fim enquanto
Calcule *média* como $\frac{soma}{contador}$
Apresente *média*

Porcentagens

Para porcentagens é preciso ter uma quantidade de ocorrências específicas e dividir esse valor pelo número total de ocorrências. Basicamente, contam-se as ocorrências específicas o e o total, realizando a divisão de um pelo outro na sequência.

Supondo uma sequência de pontuações (de 0 a 100) e desejando-se saber a porcentagem das pontuações no intervalo [90, 100], uma solução algorítmica pode ser dada pelo pseudocódigo do Algoritmo 3.11.

Algoritmo 3.11: Determinação, para uma coleção de pontuações, a porcentagem de pontuações de 90 a 100.

Descrição: Determinação da porcentagem de pontuações no intervalo [90, 100] para uma sequência de pontuações
Requer: a sequência não vazia das pontuações
Assegura: apresentação da porcentagem (de 0 a 1) de ocorrências de pontuações de 90 a 100

Inicie *contador_total* e *contador_90_100* com zero
enquanto há pontuações a serem processadas **faça**
    Obtenha *pontuação*
    Adicione 1 a *contador_total*
    se *pontuação* está no intervalo [90, 100] **então**
        Adicione 1 a *contador_90_100*
    fim se
fim enquanto
Calcule *porcentagem* como $\frac{contador\_90\_100}{contador\_total}$
Apresente *porcentagem*

Bibliografia

ESTATUTO da Criança e do Adolecente, 13 jul. 1990. https://www.planalto.gov.br/ccivil_03/leis/l8069.htm.

Índice Remissivo