**Texto base**

**3**

**Solução de Problemas - Como resolver problemas computacionais**

Gilberto Alves Pereira

***Resumo***

*Neste texto estão expostas considerações sobre uma técnica simplificada para resolver problemas computacionais salientando os principais pontos: Entendimento, Projeto, Implementação e Testes. Para cada um desses pontos são expostos seus conceitos e exemplificados com vários casos.*

**1.1. O processo de Resolução de Problemas Computacionais**

Muitos acreditam que para desenvolver um programa basta programar em alguma linguagem. Na verdade, a programação ou construção do programa é apenas uma das etapas desse processo. Antes de um programa ser construído existem passos importantíssimos que se não forem feitos podem levar a um grande prejuízo em termos de tempo e investimentos. Antes da construção precisamos pensar no projeto do programa e antes do projeto precisamos deixar bem claro o que precisa ser feito. Após o programa ser escrito, ele precisa ser testado para verificar o que atende ao que foi proposto. (DIERBACH, 2012)

**1.2 As Etapas da Resolução de Problemas Computacionais**

Podemos dividir em quatro as etapas Técnica de Solução de Problemas Computacionais:

1. Análise e Entendimento
2. Projeto
3. Construção ou Implementação
4. Testes

**1.2.1 Análise e Entendimento**

Esta é a etapa mais importante. Aqui definimos claramente o que deve ser feito, onde devemos chegar, qual o objetivo do programa. Um engano nessa etapa é desastroso. Acarretará certamente em prejuízo. Um erro aqui implicará em termos que recomeçar tudo novamente.

Para exemplificar as Etapas, vamos utilizar o seguinte problema:

Desenhar um fluxograma que receba dois números, calcule e exiba a soma desses números.

Perceba que muitas vezes nos deparamos com um problema incompleto ou mal definido. A primeira e mais importante questão no entendimento é com relação ao objetivo do problema. Está claro o objetivo desse problema? No final das contas, qual a necessidade que esse programa deverá atender?

Para deixar mais claro o entendimento vamos modelar o processo envolvido nesse caso separando as saídas, as entradas e o processamento:

* ***Saídas***: *a soma de dois números.* Essa é a informação mais importante. É o nosso objetivo. Uma dica para ajudar a encontrar essa informação, é procurar no texto palavras que tenham a ideia de saída como: exibir, apresentar, mostrar, etc.
* ***Entradas***: dois números. São os dados que precisamos usar como matéria-prima para conseguir atingir nossa saída, nosso objetivo. Uma dica para ajudar a encontrar essa informação, é procurar no texto palavras que tenham a ideia de entrada como: receba, digite, dados, etc.
* ***Processamento***: *calcule a soma*. O processamento é o que temos que fazer com as entradas para produzir as saídas. Uma dica para ajudar a encontrar essa informação, é procurar no texto verbos de ação envolvendo as entradas e saídas como: calcular, resultar, somar (aqui entra qualquer verbo que indica cálculo ou processamento), etc.

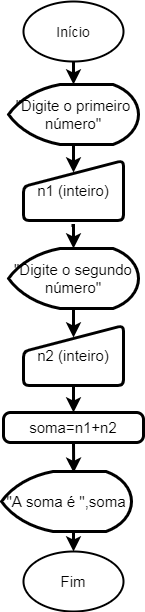
Aqui nessa etapa, é possível adiantarmos um pedaço da etapa de testes. Podemos já fazer o plano de testes, pois já sabemos as entradas e saídas esperadas. Isso será melhor visto no item de testes.

**1.2.2 Projeto**

Nesta etapa, devemos achar uma estratégia de solução de maneira a conseguir efetuar o processamento. Estamos procurando aqui um algoritmo que consiga transformar as entradas nas saídas desejadas. Devemos lembrar que um algoritmo além de envolver processos, passo-a-passo de uma solução também pode envolver uma determinada forma de armazenar os dados para conseguir efetuar o passo-a-passo. No nosso exemplo, esse algoritmo é uma simples *soma*. Apenas para deixar mais claro se o cálculo fosse uma raiz quadrada, na etapa de entendimento ficaria claro que o processamento seria uma raiz quadrada. Na etapa de projeto teríamos que buscar um algoritmo para o cálculo da raiz quadrada. Caso esse algoritmo não existisse teríamos que criar um.

**1.2.3 Implementação ou Construção**

Nesta etapa devemos definir qual ferramenta/linguagem será usada para a construção do algoritmo definido na etapa de projeto. Aqui aplicamos as ferramentas necessárias de lógica de programação para implementar o algoritmo. Essa é a etapa comumente conhecida como “programar”. No nosso exemplo, utilizaremos o fluxograma:



**Figura 3.1. Fluxograma da soma de dois números. Fonte Autor.**

Perceba que a utilização de um fluxograma permite que posteriormente ele possa ser convertido para qualquer linguagem que se queira. O Fluxograma carrega com ele toda a inteligência do entendimento e projeto de software.

**1.2.4 Testes**

Essa é uma etapa costumeiramente esquecida por muitos programadores. Não menos importante que as outras, essa etapa bem feita define a qualidade do programa. Tanto no quesito mais importante de atender ao que foi proposto atingindo seu objetivo com precisão, quanto na eficiência da maneira como foi construído.

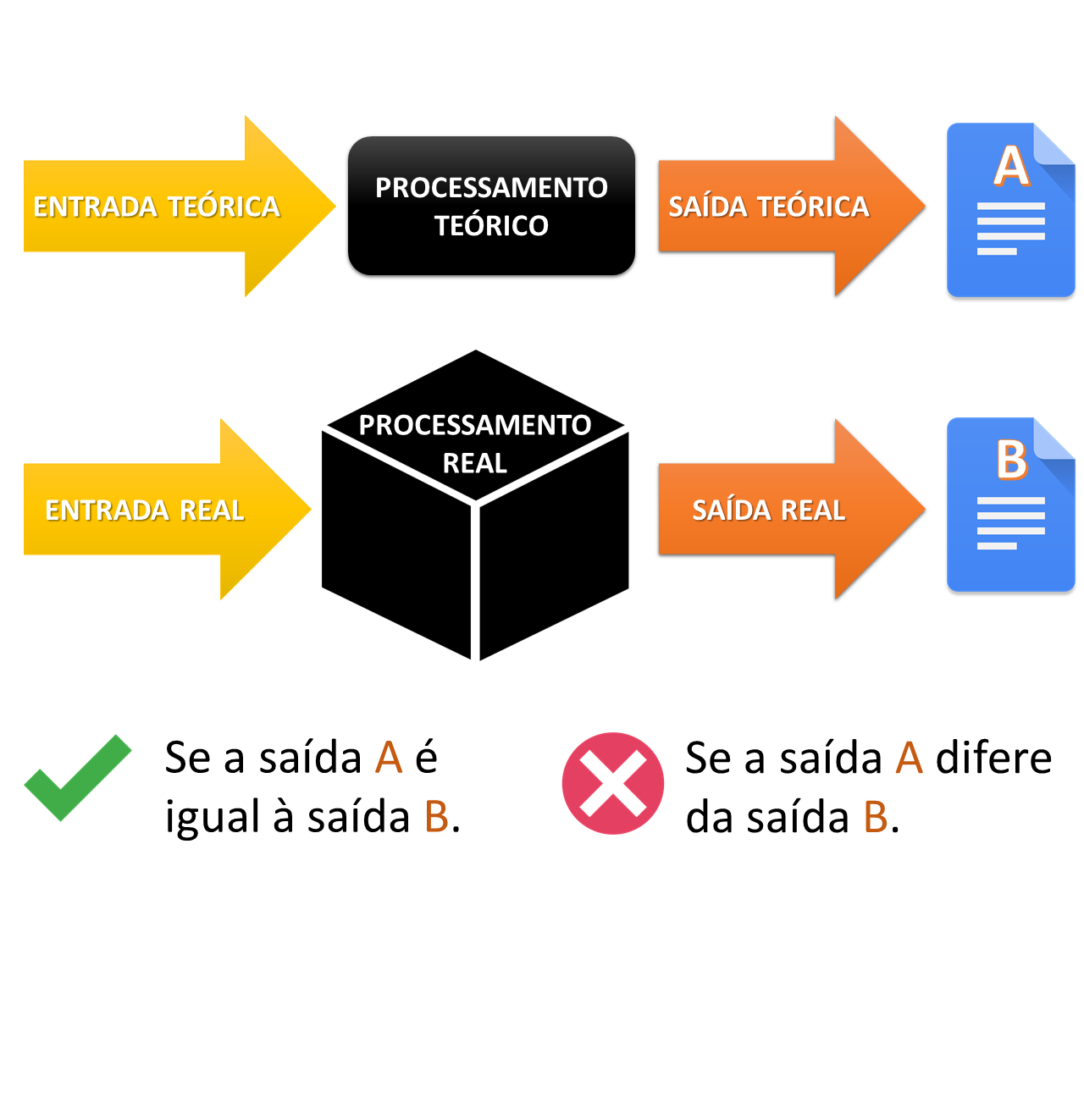
A disciplina de testes é muito abrangente, vamos mostrar aqui apenas algumas das muitas técnicas de elaboração de testes.

**1.2.4.1 Como testar um programa - Teste da caixa-preta**

Existem formas de testar a corretude de programas. Uma das formas é o teste de **caixa-preta**, cujos passos resumidamente são:

1. Escolhe-se uma entrada cuja saída correta correspondente seja conhecida, sem precisar usar o programa que será testado;
2. Simula-se a execução do fluxograma usando a entrada escolhida (TESTE DE MESA);
3. Compara-se a saída simulada com aquela teórica inicialmente esperada;
4. Caso haja diferença entre a saída teórica e a saída real, muito provavelmente o programa está incorreto e necessita de correções. As devidas modificações são feitas e retorna-se ao passo (b);
5. Caso não existam divergências entre as saídas, opta-se por: (I) encerrar os testes ou (II) submeter o programa a outros casos de teste.

Note que **testes de caixa-preta não consideram a estrutura interna** do programa, ou seja, o algoritmo usado para construir o programa, assim como a linguagem de programação em que foi implementado, não importam. Veja a ilustração simplificada do procedimento de teste de caixa-preta na Figura 3.2.



**Figura 3.2. Simplificação do funcionamento do teste de caixa-preta. Fonte Autor.**

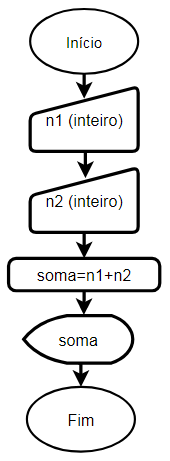
Lembre-se, **testes não garantem**, necessariamente, **que um programa está correto**. Um programa que seja aprovado em todos os casos de teste aos quais foi submetido não está obrigatoriamente correto, pois pode haver um caso de teste ausente que geraria uma falha.

Para garantir a corretude de um programa por meio de testes é necessário fazer um **teste exaustivo**. Testes exaustivos submetem o programa a todas as entradas esperadas possíveis, o que muitas vezes é impraticável. Imagine quantas possíveis entradas esperadas existem para um programa que soma dois números inteiros e exibe o resultado. Infinitas!

Uma forma de garantir a corretude de um programa sem precisar construir testes, é por meio de **prova formal**, porém esse método exige maiores conhecimentos matemáticos, técnicas avançadas de análise de algoritmos e criatividade. Provas formais costumeiramente demandam mais recursos financeiros e tempo para serem satisfatoriamente concluídas em programas mais extensos.

Geralmente, para **sistemas não críticos** (aqueles que não gerenciam laboratórios com doenças altamente contagiosas; não administram usinas nucleares; não controlam produção de foguetes; não automatizam cirurgias, aviões, trens, etc.) são montadas sequências de testes com boa abrangência, testando principalmente as extremidades das possíveis entradas esperadas. Isso amplia o nível de confiança nas soluções propostas. Em nossas aulas optamos por essa abordagem simplificada para o teste de programas.

O uso de **testes de mesa** pode auxiliar na execução de testes de caixa-branca, pois permite que toda instrução que implique em mudança nas variáveis seja representada em uma linha-coluna da tabela. No caso de testes de mesa com estruturas condicionais, é facilitador acrescentar uma coluna com a expressão da condição de seleção, sendo útil para perceber erros na definição da condição (frequentes com iniciantes).

Vamos aplicar a técnica da caixa preta ao exercício da soma de dois números, cujo fluxograma é mostrado abaixo.

**Figura 3.3. Fluxograma do algoritmo da soma de dois números. Fonte Autor.**

Vamos pensar em dois testes:

**Tabela 3.1. Testes de caixa preta da situação exemplo.**

| **Teste ID** | **Entradas** | **Saída Prevista** |
| --- | --- | --- |
| 1 | 5 e 6 | 11 |
| 2 | -10 e 5 | -5 |

Para a elaboração do teste de mesa, criamos uma tabela com as variáveis usadas e com um coluna (Tela) que mostra o que está sendo exibido na tela:

**Tabela 3.2. Variáveis e dados de saída.**

| **n1** | **n2** | **soma** | **Tela** |
| --- | --- | --- | --- |
| 5 | 6 | 11 | 11 |

Em seguida, a cada comando executado no fluxograma, registramos o valor que as variáveis envolvidas vão recebendo, conforme mostrado na tabela a seguir, na qual executamos o teste 1:

**Tabela 3.3. Atualização dos valores das variáveis.**

| Comando 1 | Teste de Mesa após a execução do comando 1   | **n1** | **n2** | **soma** | **Tela** | | --- | --- | --- | --- | | 5 |  |  |  | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Comando 2 | Teste de Mesa após a execução do comando 2   | **n1** | **n2** | **soma** | **Tela** | | --- | --- | --- | --- | | 5 | 6 |  |  | |
| Comando 3 | Teste de Mesa após a execução do comando 3   | **n1** | **n2** | **soma** | **Tela** | | --- | --- | --- | --- | | 5 | 6 | 11 |  | |
| Comando 4 | Teste de Mesa após a execução do comando 4   | **n1** | **n2** | **soma** | **Tela** | | --- | --- | --- | --- | | 5 | 6 | 11 | 11 | |

**Tabela 3.4. Resultado do primeiro teste de mesa.**

| **n1** | **n2** | **soma** | **Tela** |
| --- | --- | --- | --- |
| 5 | 6 | 11 | 11 |

**Tabela 3.5. Resultado do segundo teste de mesa.**

| **n1** | **n2** | **soma** | **Tela** |
| --- | --- | --- | --- |
| -10 | 5 | -5 | -5 |

Após a condução dos testes de mesa, podemos verificar os resultados:

**Tabela 3.6. Verificação dos resultados dos testes de mesa.**

| **Teste ID** | **Entradas** | **Saída prevista** | **Saída do teste** | **Resultado** |
| --- | --- | --- | --- | --- |
| 1 | 5 e 6 | 11 | 11 | OK |
| 2 | -10 e 5 | -5 | -5 | OK |

**1.2.4.2. Como testar um programa - Teste da caixa-branca**

Também é possível usar **testes de caixa-branca**, em que a estrutura interna do programa é avaliada, ou seja, cada instrução do algoritmo implementado em uma linguagem de programação será analisada de acordo com a sequência de execução. Neste formato de teste é necessário "abrir a caixa e ver o que está dentro".

O uso de **testes de mesa** pode auxiliar na execução de testes de caixa-branca, pois permite que toda instrução que implique em mudança nas variáveis seja representada em uma linha-coluna da tabela. No caso de testes de mesa com estruturas condicionais, é facilitador acrescentar uma coluna com a expressão da condição de seleção, sendo útil para perceber erros na definição da condição (frequentes com iniciantes).

Bons testes de caixa-branca para programas com estruturas de seleção buscam passar por **todos os caminhos do algoritmo**, garantindo que sejam executadas e analisadas tanto as instruções do "bloco do caminho se verdadeiro" quanto do "bloco do caminho se falso", se houver.

Pense no seguinte **exemplo**: crie um fluxograma que leia como entrada dois salários distintos e exiba uma mensagem com o valor do maior deles, acrescido de 20% de bônus. Vamos construir um fluxograma, fazer dois testes de caixa-preta (sem comparação com as saídas de um programa real, pois não o codificamos) e dois testes de caixa-branca.

| Fluxograma | Entrada e saída do 1º teste    Entrada e saída do 2º teste |
| --- | --- |

**Figura 3.4. Fluxograma de entradas e saídas dos testes. Fonte Autor.**

**Tabela 3.7. Primeiro teste de mesa**

| **s1** | **s2** | **maior** | **s1 > s2** | **Tela** |
| --- | --- | --- | --- | --- |
| 10000.00 | 5000.00 | 10000.00 | 10000.00 > 5000.00 (True) | 12000.00 |

**Tabela 3.8. Segundo teste de mesa**

| **s1** | **s2** | **maior** | **s1 > s2** | **Tela** |
| --- | --- | --- | --- | --- |
| 2500.77 | 3500.25 | 3500.25 | 2500.77 > 3500.25 (False) | 4200.30 |

Como podemos perceber, os resultados produzidos pelo fluxograma, analisados com auxílio dos testes de mesa, são iguais àqueles previstos no teste de caixa-preta. Portanto, há maior confiança que o algoritmo faz o que deve fazer, ou seja, existem mais indícios de que esteja correto.

**1.3. Referências**

DIERBACH, C. “Introduction to Computer Science Using Python: A Computational Problem-Solving Focus”1st Edition, New York: Wiley, 2012.