Introdução a Algoritmos

Raoni F. S. Teixeira

Aula 1 - Introdução

1 Introdução

Programar é o ato de instruir um computador a realizar uma tarefa. Para isso, precisamos compreender tanto o que é um computador quanto como ele processa instruções. Nesta primeira aula, exploraremos os elementos fundamentais que formam o ambiente computacional e o papel dos algoritmos na resolução de problemas.

2 O que é um Computador

A palavra *computador* vem de *computar*, que significa calcular. Antes das máquinas modernas, os "computadores" eram pessoas responsáveis por executar cálculos repetitivos, seguindo instruções precisas.

Hoje, um computador é uma máquina capaz de realizar, em alta velocidade, operações matemáticas e lógicas sobre dados de entrada, produzindo resultados úteis. Sua essência é o processamento de informações, mediado por dois componentes centrais: **hardware** e **software**.

3 Hardware e Software

O **hardware** é a parte física do computador — processador, memória, dispositivos de entrada e saída. Já o **software** é o conjunto de instruções que dá vida ao hardware. Essas instruções são expressas em linguagens formais que o processador interpreta e executa.

Internamente, a linguagem do computador é binária. Tudo é representado por combinações de zeros e uns (bits). O agrupamento de 8 bits forma um **byte**, unidade capaz de representar um caractere, número ou símbolo.

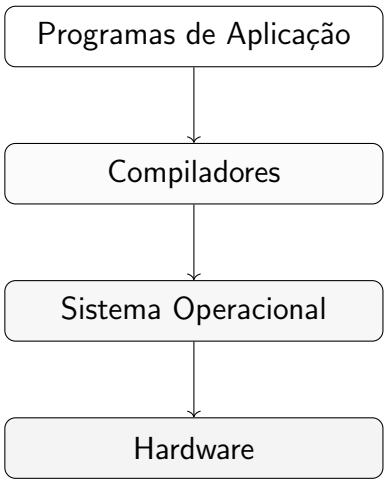

Figura 1: Hierarquia de abstrações em um sistema computacional.

Cada camada esconde detalhes da inferior, simplificando o controle do sistema.

4 Algoritmos

Um **algoritmo** é uma receita: uma sequência de passos claros para resolver um problema. Ele é a base da programação, pois define a lógica de uma solução antes mesmo de se escrever o código.

Imagine, por exemplo, multiplicar dois números grandes, como 2345×4567, usando apenas lápis e papel. O método que você segue—multiplicar dígito por dígito e somar os resultados—é um algoritmo.

Um bom algoritmo é claro, preciso e pode ser executado por uma máquina. Criar um é como montar um quebra-cabeça: as peças (as ações possíveis) são conhecidas; o desafio é encaixá-las na ordem correta.

Do Problema à Solução

Um problema computacional define uma relação entre uma entrada e uma saída. Por exemplo, o problema de fatorar um número tem como entrada um número inteiro n e como saída um de seus fatores primos.

Para construir um algoritmo, podemos seguir uma estratégia de quatro etapas:

1. **Compreender o Problema:** É impossível resolver o que não se entende.
2. **Elaborar um Plano:** Buscamos conexões entre os dados e a solução. Muitas vezes, a resposta surge ao resolvermos um problema mais simples e relacionado. O resultado desta etapa é o esboço do algoritmo.

3. **Executar o Plano:** Colocamos o algoritmo em ação, passo a passo.

4. **Avaliar a Solução:** Verificamos se a resposta faz sentido. Se não, voltamos ao início.

Esse processo exige curiosidade e trabalho duro. Anote tudo. No começo, tudo parece complexo, então aja como um aprendiz de cozinha: no início, siga as receitas à risca; com a prática, você entenderá os princípios e começará a criar suas próprias soluções.

5 Da Solução aos Programas

Para que um computador entenda um algoritmo, é preciso traduzi-lo para uma **linguagem de programação**. Essa linguagem deve ser expressiva o bastante para representar qualquer cálculo, mas suficientemente rigorosa para evitar ambiguidades.

Historicamente, as primeiras linguagens eram extremamente próximas do hardware, compostas apenas por números binários — o chamado **código de máquina**. Em seguida surgiu o **Assembly**, que substituiu números por mnemônicos mais legíveis:

```
LOOP: MOV A, 3
INC A
JMP LOOP
```

Mais tarde, apareceram as **linguagens de alto nível**, como C, Pascal e Java. Elas aproximam a programação da linguagem humana, facilitando a expressão de ideias complexas.

6 A Linguagem C

A linguagem **C** é uma das mais influentes da história da computação. Criada na década de 1970 por Dennis Ritchie, tornou-se a base para o desenvolvimento de sistemas operacionais, compiladores e linguagens modernas.

Um programa em C é escrito em um arquivo texto (código-fonte) e segue uma estrutura padrão:

```
#include <stdio.h>

int main() {
    printf("Hello, world!\n");
    return 0;
}
```

Para executá-lo, é necessário **compilá-lo**, transformando o código-fonte em um arquivo executável:

```
$ gcc hello.c -o hello
$ ./hello
Hello, world!
```

7 Erros e Depuração

Durante o desenvolvimento, é comum encontrar erros de **compilação** (violação das regras da linguagem) e **erros de execução** (comportamentos inesperados). Aprender a ler mensagens de erro é parte essencial da prática da programação.

Ferramentas como **depuradores (debuggers)** permitem executar o programa passo a passo, inspecionar variáveis e identificar falhas lógicas. Exemplo: o comando `gdb` em sistemas Unix.

8 Exemplo Prático

O programa abaixo lê dois números e exibe o maior deles:

```
#include <stdio.h>
int main() {
    int x, y;
    printf("x: ");
    scanf("%d", &x);
    printf("y: ");
    scanf("%d", &y);
    if (x > y)
        printf("O maior número é x = %d\n", x);
    else
        printf("O maior número é y = %d\n", y);
}
```

Esse exemplo mostra como uma tarefa simples — comparar dois valores — é descrita passo a passo em C. O mesmo raciocínio se aplica a problemas mais complexos, bastando decompor o raciocínio em etapas precisas.

9 Conclusão

Nesta aula, vimos que programar é construir pontes entre **ideias** e **máquinas**. Os computadores executam operações simples, mas sua combinação permite resolver problemas sofisticados. O domínio da programação começa pela compreensão dos fundamentos — hardware, software, algoritmos e linguagens formais —, sobre os quais construiremos aplicações cada vez mais elaboradas ao longo do curso.

Referências

- Ritchie, D. M. & Kernighan, B. W. *The C Programming Language.* Prentice Hall, 1978.