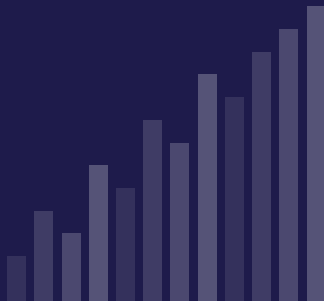

AULA 1 · BLOCO 1: FUNDAMENTOS, ORDENAÇÃO E CUSTO

Algoritmos, eficiência e o método experimental

Capítulo 1

Estruturas de dados e seus algoritmos · José Dihego · IFBA



Roteiro da aula

Ao final, você será capaz de

- ▶ Definir algoritmo, instância e corretude
- ▶ Explicar por que eficiência importa mais que hardware
- ▶ Medir tempo de execução em C de forma confiável
- ▶ Usar a razão de dobra para reconhecer crescimento linear e quadrático



Teoria

Algoritmos como tecnologia; o exemplo dos dois computadores



Prática em C

Cronômetro em C: relógio monotônico, repetições, melhor de 5



Experimento

Dobrar n e observar o tempo: $\times 2$ ou $\times 4$?



Conexões

IA, criptografia, computação verde e engenharia de desempenho

O que é um algoritmo

- ▶ Um procedimento computacional bem definido que recebe uma **entrada** e produz uma **saída**.
- ▶ Resolve um **problema**: a relação desejada entre entrada e saída. Uma **instância** é uma entrada concreta.
- ▶ É **correto** se, para toda instância, para com a saída certa.
- ▶ Uma **estrutura de dados** organiza os dados para facilitar acesso e modificação: nenhuma serve para tudo.
- ▶ Alguns problemas não têm solução eficiente conhecida: os **NP-completos** (caixeiro-viajante, por exemplo).

Este curso: algoritmos clássicos de ordenação, busca e grafos e as estruturas que os tornam rápidos, sempre com teoria e medição.

Eficiência vale mais que hardware

Ordenar $n = 10^7$ números (exemplo da seção 1.2):

	Computador A	Computador B
Velocidade	10^{10} instr./s	10^7 instr./s (1000× mais lento)
Algoritmo	inserção, $2n^2$	merge sort, $50 n \lg n$
Programador	o melhor do mundo, em assembly	mediano, com compilador ruim
Tempo	$\frac{2 \cdot (10^7)^2}{10^{10}} = 20\,000 \text{ s} \approx \mathbf{5,5 \text{ h}}$	$\frac{50 \cdot 10^7 \lg 10^7}{10^7} \approx 1\,163 \text{ s} < \mathbf{20 \text{ min}}$

O computador 1000 vezes mais lento termina **17 vezes antes**. Com $n = 10^8$, a vantagem passa de 100 vezes.

Duas maneiras de medir

Contar operações

Conta comparações, atribuições ou acessos.

- ▶ Independe da máquina e do compilador.
- ▶ Exata e reproduzível.
- ▶ Ignora cache, desvio de fluxo e memória.

Medir tempo

Relógio antes e depois.

- ▶ Mostra o custo real.
- ▶ Ruidoso: outros processos, frequência da CPU.
- ▶ Pede repetições e o **mínimo** (ou a mediana).

Razão de dobra: se $T(n) \approx cn^k$, então $T(2n)/T(n) \approx 2^k$. Razão 2: linear. Razão 4: quadrático. Razão 8: cúbico.



Prática em C

- ▶ `clock_gettime(CLOCK_MONOTONIC)`: o relógio certo para medir
- ▶ Um algoritmo linear (soma) e um quadrático (pares com soma zero)
- ▶ `volatile` impede o compilador de apagar o trabalho

Armadilhas de quem mede tempo

- ▶ **Otimização elimina código morto.** Se o resultado não é usado, -O2 pode apagar o laço inteiro. Guarde o resultado numa variável `volatile`.
- ▶ **Resolução do relógio.** `clock()` mede tempo de CPU em passos grosseiros; use `clock_gettime` e repita até o tempo passar de milissegundos.
- ▶ **Ruído do sistema.** Interrupções só aumentam o tempo: o **mínimo** de várias execuções é a estimativa mais limpa.
- ▶ **Aquecimento.** A primeira execução paga falhas de página e cache frio.

Cronômetro e dois algoritmos (1/3)

```
1  #define _POSIX_C_SOURCE 199309L    // clock_gettime com -std=c17
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <time.h>
5
6  /* Tempo de relógio em segundos (relógio monotônico: não volta com ajustes de hora). */
7  double agora(void) {
8      struct timespec t;
9      clock_gettime(CLOCK_MONOTONIC, &t);
10     return t.tv_sec + t.tv_nsec / 1e9;
11 }
```

Cronômetro e dois algoritmos (2/3)

```
13 long long soma(const int A[], int n) {           // uma passada: cresce como n
14     long long s = 0;
15     for (int i = 0; i < n; i++) s += A[i];
16     return s;
17 }
18
19 long pares_zero(const int A[], int n) {           // todos os pares i < j: cresce como n^2
20     long cont = 0;
21     for (int i = 0; i < n; i++)
22         for (int j = i + 1; j < n; j++)
23             if (A[i] + A[j] == 0) cont++;
24     return cont;
25 }
```

Cronômetro e dois algoritmos (3/3)

```
27 int main(void) {
28     int n;
29     if (scanf("%d", &n) != 1 || n <= 0) return 1;
30     int *A = malloc(n * sizeof(int));
31     for (int k = 0; k < n; k++)
32         if (scanf("%d", &A[k]) != 1) return 1;
33     double t0 = agora();
34     printf("soma = %lld\n", soma(A, n));
35     printf("pares com soma zero = %ld\n", pares_zero(A, n));
36     printf("tempo: %.6f s\n", agora() - t0);
37     free(A);
38     return 0;
39 }
```



Experimento

- ▶ Hipótese: $T_{\text{lin}} \approx a n$ e $T_{\text{quad}} \approx b n^2$
- ▶ Melhor de 5 execuções, n de 2.000 a 32.000
- ▶ Ajuste da constante por mínimos quadrados

Dobrar n : o tempo dobra ou quadruplica?

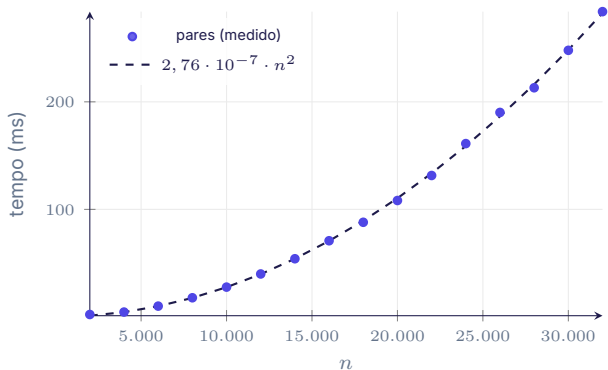
Hipótese. A soma percorre o vetor uma vez: $T(n) \approx an$. A contagem de pares examina $\binom{n}{2}$ pares: $T(n) \approx bn^2$. Ao dobrar n , esperamos razões perto de 2 e de 4. As constantes a e b dependem da máquina; as **formas** não.

Método

1. Vetor aleatório de inteiros em $[-1000, 1000]$
2. n de 2.000 a 32.000, passo 2.000
3. Tempo: mínimo de 5 execuções (CLOCK_MONOTONIC)
4. Ajuste $c \cdot f(n)$ pelo método dos mínimos quadrados

🧪 EXPERIMENTO

Tempo do algoritmo quadrático



n	medido	razão
2.000	2,222	$5,55 \cdot 10^{-7}$
8.000	17,8	$2,78 \cdot 10^{-7}$
14.000	54,1	$2,76 \cdot 10^{-7}$
20.000	108,2	$2,71 \cdot 10^{-7}$
26.000	190,1	$2,81 \cdot 10^{-7}$
32.000	283,9	$2,77 \cdot 10^{-7}$

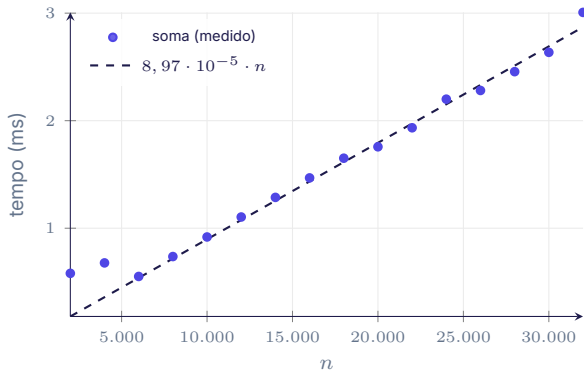
razão = medido $\div$ n^2

✓ desvio máx. 2,0%

A razão tempo/ n^2 estabiliza: esse valor é a constante b da sua máquina.

🧪 EXPERIMENTO

Tempo do algoritmo linear



n	medido	razão
2.000	0,580	$2,90 \cdot 10^{-4}$
8.000	0,737	$9,21 \cdot 10^{-5}$
14.000	1,287	$9,19 \cdot 10^{-5}$
20.000	1,757	$8,79 \cdot 10^{-5}$
26.000	2,281	$8,77 \cdot 10^{-5}$
32.000	3,007	$9,40 \cdot 10^{-5}$

razão = medido $\div$ n

✓ desvio máx. 4,8%

Uma reta. Pontos fora da curva são interrupções do sistema: por isso usamos o mínimo de 5.

EXPERIMENTO

O que o experimento mostrou

Os dois tempos seguem a forma prevista: a curva do quadrático é uma parábola e a do linear, uma reta. Para $n = 32.000$, o algoritmo quadrático já gasta cerca de 100 vezes mais que o linear, e a distância cresce com n . **A forma da curva é uma propriedade do algoritmo; a constante é da máquina.**

Rode na sua máquina: as constantes mudam, as razões de dobra não.

➤ Reproduza: `gcc -O2 aula01_experimento.c -o exp -lm && ./exp`



Conexões

- ▶ Inteligência artificial
- ▶ Criptografia
- ▶ Computação verde
- ▶ Engenharia de desempenho

Onde isso aparece na computação

Inteligência artificial

Treinar um modelo de linguagem custa da ordem de $6 \times \text{parâmetros} \times \text{tokens}$ operações: a análise de custo decide quais modelos são viáveis.

Criptografia

A segurança do RSA depende de não existir algoritmo eficiente conhecido para fatorar inteiros grandes: a ineficiência vira proteção.

Computação verde

Data centers consomem cerca de 1–2% da energia elétrica mundial. Trocar n^2 por $n \lg n$ economiza energia, não só tempo.

Engenharia de desempenho

Ferramentas como perf, gprof e testes de regressão de desempenho em integração contínua aplicam o método desta aula na indústria.

Para casa

Exercícios

1. (1.1-1) Dê um exemplo do mundo real que exija ordenação e um que exija o menor caminho.
2. (1.2-2) Para que n a inserção ($8n^2$ passos) vence o merge sort ($64n \lg n$ passos)?
3. (1.2-3) Qual o menor n em que $100n^2$ é mais rápido que 2^n ?
4. Rode o experimento com -00 e com -02. O que muda: a forma ou a constante?
5. Implemente uma contagem de pares em $O(n \lg n)$ (ordene e use dois índices) e meça.

Leitura

Cormen et al., capítulo 1 e problema 1-1.

Código

josedihego.net/estruturas