Sobre a Complexidade dos Algoritmos

Laboratório de Programação Competitiva I - 2022

Pedro Henrique Paiola
Rene Pegoraro
Wilson M Yonezawa
Unesp Bauru

Introdução

- O que é um algoritmo?
- Objetivo: Encontrar e/ou projetar algoritmos eficientes para **problemas** computacionais
- Como projetar e avaliar um algoritmo?

> “Informalmente, um **algoritmo** é qualquer procedimento computacional bem definido que recebe algum valor, ou conjunto de valores, como entrada e produz algum valor, ou conjunto de valores, como saída.
> Um algoritmo é, portanto, **uma sequência de passos computacionais que transformam a entrada na saída**”. Tradução livre do livro “Introduction to Algorithms do Cormen, Leiserson, Rivest e Stein, 2009.

Algoritmo (exemplo de representação)

mdc(a, b)

Etapa 1: Dados dos números inteiros a e b

Etapa 2: R é o resto da divisão de a por b

Etapa 3: Faça a = b e b = R

Etapa 4: Repita as etapas 2 e 3 enquanto o resto de a dividido por b for maior que zero

Etapa 5: mdc = b

Etapa 6: Fim

Recursiva

$$mdc(a,b) = \begin{cases} a & \text{se } b = 0 \\ mdc(a,\ a \bmod b) \end{cases}$$

Eficiência de um algoritmo

- Tempo (de execução) e espaço (memória)
- Tempo de execução pode variar dependendo do computador
- Comportamento do algoritmo em função do tamanho da entrada de dados (n)

Eficiência de um algoritmo (exemplo)

Métodos para solução de equações lineares

tabela 1.1.1 Tamanho do problema × tempo de execução

n	Método de Cramer	Método de Gauss
2	22 µs	50 µs
3	102 µs	150 µs
4	456 µs	353 µs
5	2,35 ms	666 µs
10	1,19 min.	4,95 ms
20	15225 séculos	38,63 ms
40	5.10^{33} séculos	0,315 s

Fonte: Complexidade de Algoritmos - V13 - UFRGS. Toscani e Velos, 2012.

Exemplos de problemas

- Encontrar o maior elemento de uma lista de tamanho N
- Mostrar todos os pares ordenados de uma lista de tamanho N
- Ordenar uma lista com N números inteiros
- Encontrar um elemento em uma lista ordenada de tamanho N
- Permutar as letras de uma string de tamanho N
- 3-SAT, problema do caixeiro viajante, etc

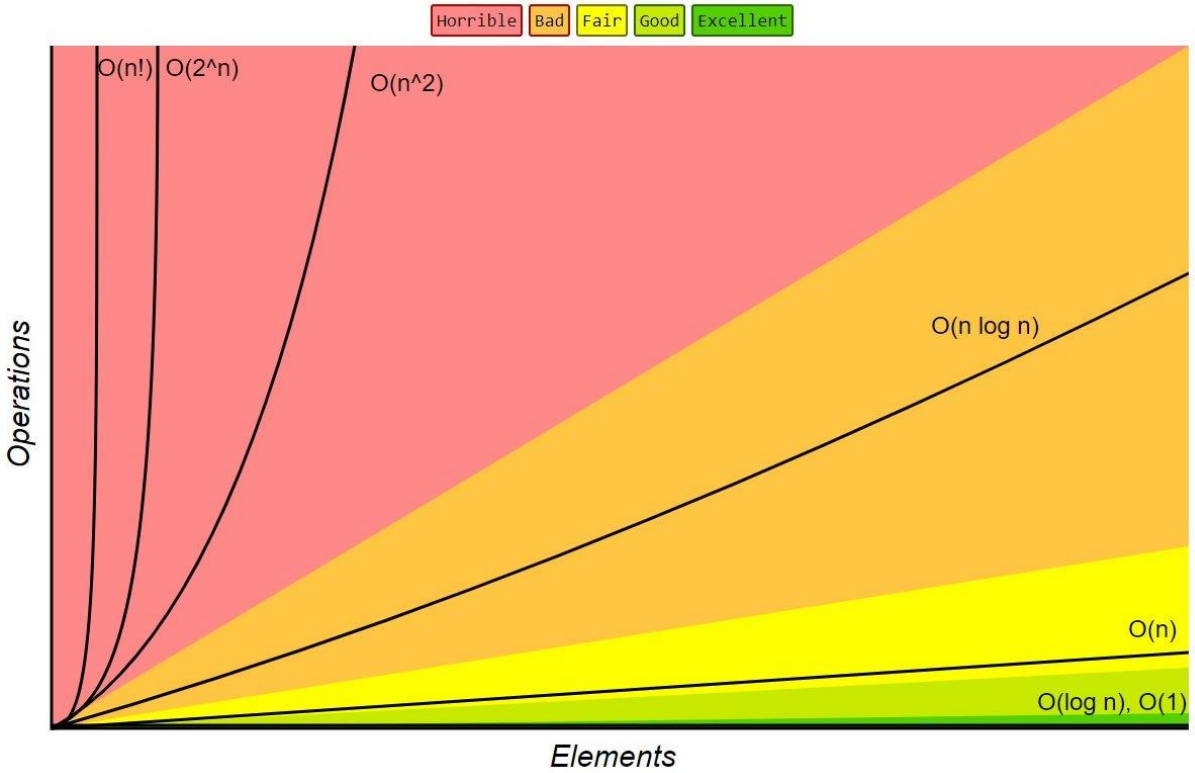
Big O

A notação Big O descreve a **complexidade** de uma função/algoritmo, seja seu **tempo de execução** (a quantidade de tempo que um algoritmo leva para completar sua tarefa) ou a **complexidade do espaço** (a quantidade de espaço que um algoritmo usa). É usado para ver quanto mais trabalho precisa ser feito por um algoritmo **quando a entrada aumenta**

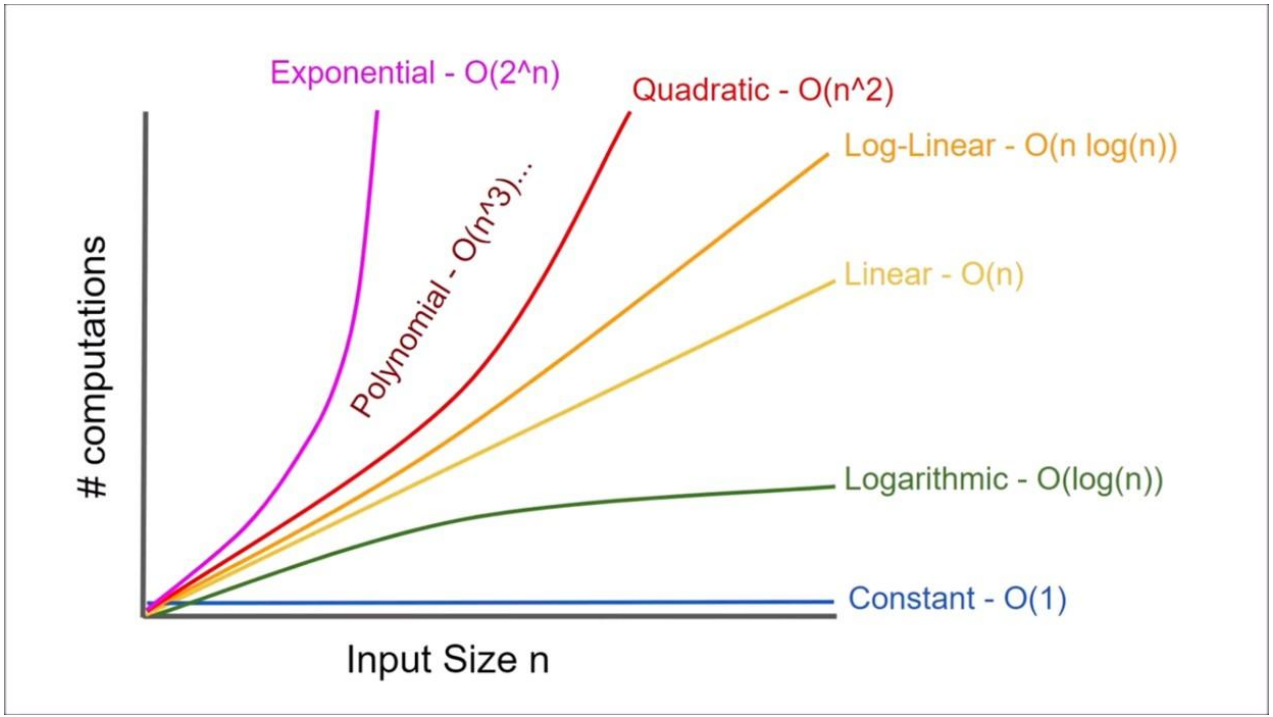
Complexidade do algoritmo

Complexidade	Terminologia
O(1)	Complexidade constante
O(log n)	Complexidade logarítmica
O(n)	Complexidade linear
O(n log n)	Complexidade n log n
$O(n^b)$	Complexidade polinomial
$O(b^n)$, $b > 1$	Complexidade exponencial
O(n!)	Complexidade fatorial

Complexidade do algoritmo



Complexidade do algoritmo



Fonte: https://www.youtube.com/watch?v=47GRtdHOKMg

Tempo computacional (funções polinomiais e exponenciais)

Função / *Complexidade*	*Tamanho n* 10	20	30	40	50	60
n	0,00001 segundo	0,00002 segundo	0,00003 segundo	0,00004 segundo	0,00005 segundo	0,00006 segundo
n^2	0,0001 segundo	0,0004 segundo	0,0009 segundo	0,0016 segundo	0,0025 segundo	0,0036 segundo
n^3	0,001 segundo	0,008 segundo	0,027 segundo	0,064 segundo	0,125 segundo	0,216 segundo
n^5	0,1 segundo	3,2 segundos	24,3 segundos	1,7 minuto	5,2 minutos	13,0 minutos
2^n	0,001 segundo	1,0 segundo	17,9 minutos	12,7 dias	35,7 anos	366 séculos
3^n	0,059 segundo	58 minutos	6,5 anos	3855 séculos	2×10^8 séculos	$1,3 \times 10^{13}$ séculos
$n!$	3,628 segundos	771,4 séculos	$8,4 \times 10^6$ séculos	$2,5 \times 10^{32}$ séculos	$9,6 \times 10^8$ séculos	$2,6 \times 10^{66}$ séculos

Obs: Considerando um computador que executa 10^6 operações por segundo

Fonte: Pesquisa Operacional para cursos de engenharia. Arenales et al, 2017.

Voltando aos Exemplos

- O(n) - Encontrar o maior elemento de uma lista de tamanho N
- $O(n^2)$ - Mostrar todos os pares ordenados de uma lista de tamanho N
- O (log n) - Encontrar um elemento em uma lista ordenada de tamanho N
- O(n log n) - Ordenar uma lista com N números inteiros
- $O(k^n)$ - 3-SAT, problema do caixeiro viajante, etc
- O(n!) - Permutar as letras de uma string de tamanho N

Voltando aos Algoritmos

mdc(a, b)

Etapa 1: Dados dos números inteiros a e b

Etapa 2: R é o resto da divisão de a por b

Etapa 3: Faça a = b e b = R

Etapa 4: Repita as etapas 2 e 3 enquanto o resto de a dividido por b for maior que zero

Etapa 5: mdc = b

Etapa 6: Fim

```
int mdc(int a,int b) {
  int R;
  while ((a % b) > 0)  {
    R = a % b;
    a = b;
    b = R;
  }
  return b;
}
```

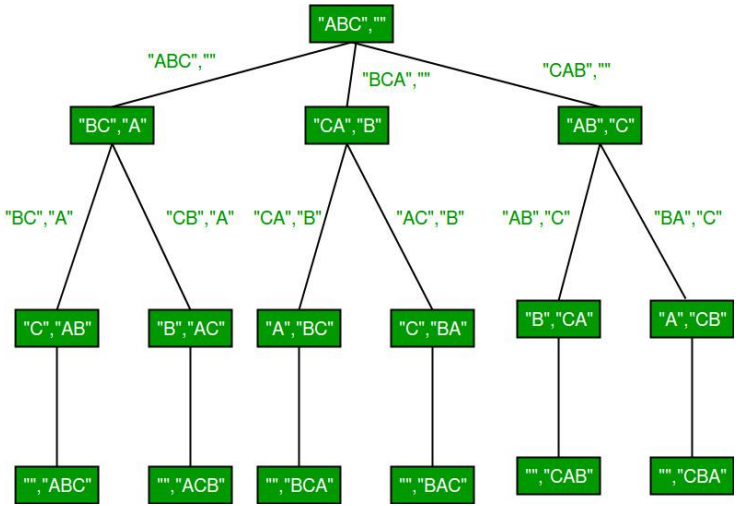
Voltando aos Algoritmos

$$mdc(a,b) = \begin{cases} a & \text{se } b = 0 \\ mdc(a, a \bmod b) \end{cases}$$

```
int mdc(int a, int b)
{
    if (a == 0)
       return b;
    if (b == 0)
       return a;
    // caso base
    if (a == b)
        return a;
    if (a > b)
        return mdc(a-b, b);
    return mdc(a, b-a);
}
```

Voltando aos Algoritmos

- O(3!) - Permutar as letras de uma string de tamanho 3



Fonte: https://www.geeksforgeeks.org/permutations-of-a-given-string-using-stl/

Considerações finais

Observar o problema e o tamanho da entrada (N)

Procurar e/ou buscar projetar algoritmos com complexidade: **logarítmica O(log N), linear O(n), linear/logarítmica O(n log n), polinomial $O(n^k)$.**

Se a complexidade do problema é do tipo exponencial ou fatorial, verificar se é possível reduzir para polinomial.

Problemas recursivos devem ser analisados com cuidado visto que, geralmente, aumentam a complexidade do algoritmo.