Aula 1

Estrutura de Dados

Prof. Vinicius Pozzobon Borin

1

Conversa Inicial

2

- Estruturas de dados são maneiras de organizá-los e colecioná-los
- A forma como os dados ficam organizados dentro da memória, o acesso a eles e suas manipulações caracterizam a nossa disciplina

3

- Python: listas, dicionários e tuplas
- C/C++/Java: vetor/array, struct, map

4

- O objetivo desta aula é introduzir os principais conceitos inerentes à complexidade de algoritmos
- Tais conceitos serão recorrentes ao longo das próximas aulas
- Veremos a complexidade Big-O e suas implicações aos algoritmos

5

Pesquisa em um conjunto de dados

6

- Será que todo algoritmo que resolve o mesmo problema apresenta igual desempenho?
  - Vamos investigar!

7

Pesquisa sequencial

- Jack quer que Joana adivinhe qual número ele está pensando, de 1 até 100
- Jack se compromete a dizer para Joana se o valor que ele pensou é maior ou menor

ivector/Shutterstock

Volha Hlinskaya/Shutterstock

8

- Joana decidiu tentar um número por vez, iniciando do 1 para cima
- Jack pensou no número 99

$\boxed{99} \neq \boxed{99}$

9

- Foram 99 tentativas até Joana acertar o valor!
- Vamos analisar este problema em um código?

10

Pesquisa binária

- Joana teve uma ideia: "E se eu for sempre chutando o valor do meio?"

Volha Hlinskaya/Shutterstock

11

$\boxed{99} \neq \boxed{99}$

12

- 7 tentativas
- O algoritmo usado sempre quebra o conjunto de dados ao meio
- Vejamos sua implementação em Python

13

- E se o número a ser adivinhado fosse 1?
- A busca sequencial não se sairia melhor do que a binária?
  - Sim, por isso a ordem de organização dos dados no conjunto impacta o desempenho do algoritmo

14

Análise de algoritmos

15

- Como podemos comparar o desempenho de diferentes algoritmos para uma mesma aplicação? Podemos mensurar isso?

16

- Um algoritmo mais eficiente para resolver um problema é um algoritmo de menor complexidade
- A complexidade do algoritmo cresce à medida que o tamanho do conjunto de dados (n) também cresce

17

Complexidade de algoritmos

- Complexidade de tempo: a quantidade de tempo que um algoritmo leva para completar sua execução
  - A quantidade de instruções do código impacta diretamente este desempenho
- ~~Complexidade de espaço: a quantidade de memória requerida para um algoritmo executar~~
  - ~~A quantidade de variáveis e seus tamanhos impactam diretamente este desempenho~~

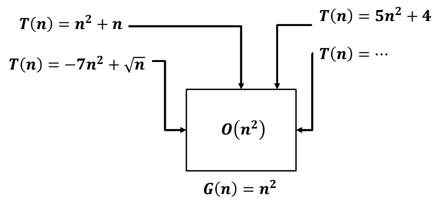
18

Complexidade de tempo

- **Tamanho do conjunto de dados de dados de entrada (n): quanto mais dados temos para manipular, mais tempo**
- **Disposição dos dados dentro do conjunto: a ordem com que os dados estão organizados no conjunto implica distintas situações**

19

Notação Big-O



20

- **A notação nos dá a tendência de funcionamento de um algoritmo. A ordem de grandeza**
- **Para sabermos a complexidade de um algoritmo utilizando a notação Big-O, basta encontrarmos o termo de maior grau da equação que o descreve**

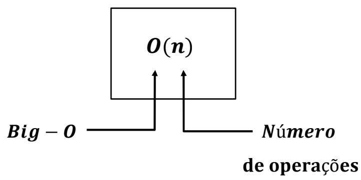
21

Tipos de notação

- **Big-O: pior caso (limite inferior)**
- **Big-Ω: melhor caso (limite superior)**
- **Big-θ: caso médio**

22

Notação Big-O



23

Exemplo

- **Um algoritmo realiza somatórios**
- **Se cada somatório levar 1ms, e n = 10**

$$O(n) = 10 * 1ms = 10ms$$

$$O(n^2) = 10^2 * 1ms = 100ms$$

24

Encontrando o Big-O de algoritmos

25

- **Como efetivamente encontramos os Big-O dos algoritmos?**

26

Algoritmo sem laço

- **O(1)**

```
x = input('Digite um valor')
y = input('Digite um valor')

print(x + y)
```

27

Laço simples

- **O(n)**

```
def boo(dados):
  for i in dados:
    print(i)
```

28

Propriedade da adição

```
def boo(dadosA, dadosB):
  for i in dadosA:
    print(i)
  for i in dadosB:
    print(i)
```

29

$$T(n) = T_{laço1}(n) + T_{laço2}(n) = O(n) + O(n) = O(2n)$$

$$T(n) = O(n)$$

30

Propriedade da multiplicação

```
def boo(dadosA, dadosB):
  for i in dadosA:
    for j in dadosB:
      print(i + j)
```

31

Propriedade da multiplicação

$$T(n) = T_{laço1}(n) * T_{laço2}(n) = O(n) * O(n) = O(n^2)$$
$$T(n) = O(n^2)$$

32

Progressão aritmética (PA)

- **Uma sequência de números na qual a diferença entre dois termos consecutivos é constante**

$$3 + 5 + 7 + 9 + 1 \ldots + n$$
$$8 + 4 + 0 - 4 - 8 \ldots + n$$

33

$$S_n = \frac{(a_1 + a_n).n}{2}$$
$$1 + 2 + 3 + 4 + \cdots + n = \frac{(1+n).n}{2} = \frac{n^2}{2} + \frac{n}{2}$$
$$S_n = \frac{n^2}{2} + \frac{n}{2} = O(n^2)$$

34

```
def boo(dadosA, dadosB):
  for i in dadosA:
    for j in dadosB:
      print(i, j)
```

35

- **PA constante**

$$10 + 10 + 10 + \cdots + \cdots$$

36

Progressão geométrica (PG)

- **É uma sequência de números na qual existe uma razão entre um número e o seu sucessor**

$$\mathbf{1+3+9+27+\cdots+n}$$

$$\mathbf{1+2+4+8+16\ldots+n}$$

37

$$S_n = \frac{a_1.(q^n-1)}{q-1} \quad q = {a_n}/{a_{n-1}}$$

$$q = \frac{a_n}{a_{n-1}} = \frac{2}{1} = 2$$

$$\frac{a_1.(q^n-1)}{q-1} = \frac{1.(2^n-1)}{2-1} = 2^n - 1 = O(2^n)$$

38

```
def boo(dadosA, dadosB):
  for i in dadosA:
    for j in range(0, j < i * i, 1):
      print(i, j)
```

39

- **Variável j depende de i**
  - `j < i * i`

i	j
0	0
1	1
2	4
3	9
4	16

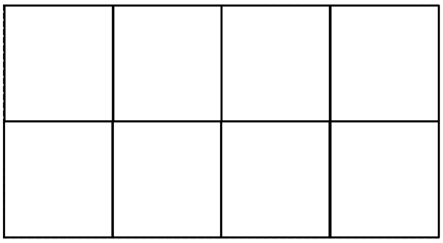
40

Dividir para conquistar

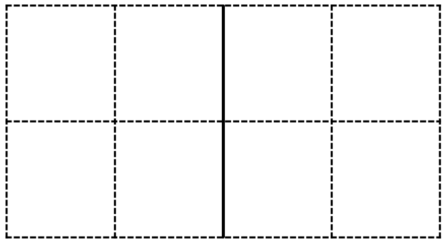
41

- **Técnica para resolver problemas**
- **Um problema muito complexo pode ser dividido em partes menores**

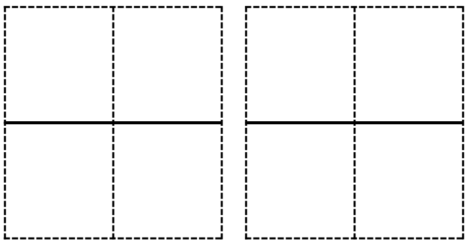
42



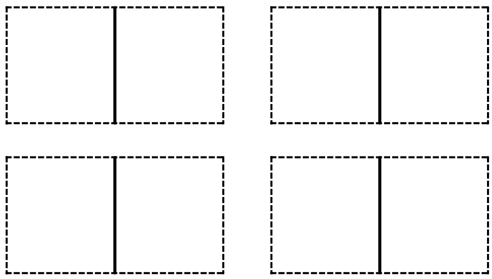
43



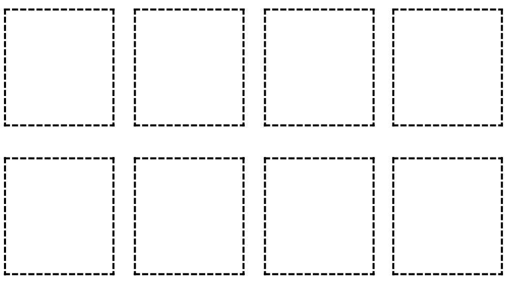
44



45



46



47

- **Dividimos um algoritmo recursivamente**
- **Duas etapas:**
  - **1. Descobrir o caso-base, que será sempre a menor parte possível para o problema**
  - **2. Descobrir como reduzir o problema para que ele se torne o caso-base**

48

Busca binária

- **Conjunto de dados (n); tentativas (k)**

$$\left.\begin{matrix}16\\8\\4\\2\\1\end{matrix}\right\} n = 16; k = 5 \qquad \left.\begin{matrix}8\\4\\2\\1\end{matrix}\right\} n = 8; k = 4$$

$$\left.\begin{matrix}4\\2\\1\end{matrix}\right\} n = 4; k = 2 \qquad \left.\begin{matrix}2\\1\end{matrix}\right\} n = 2; k = 1 \qquad 1\} n = 1; k = 1$$

49

- **Conjunto de dados (n); tentativas (k)**

k	n	n como potência de 2
5	16	$2^4 = 2^5 - 1$
4	8	$2^3 = 2^4 - 1$
3	4	$2^2 = 2^3 - 1$
2	2	$2^1 = 2^2 - 1$
1	1	$2^0 = 2^1 - 1$

50

$$2^{k-1} = n$$

$$k - 1 = \log_2 n$$

$$k = \log_2 n + 1$$

$$O(\log n)$$

51

Complexidade da recursividade

52

Exemplo: fatorial

$$5! = 5 * 4 * 3 * 2 * 1 = 120$$

$$5! = 5 * 4 * 3 * 2 * 1!$$

$$5! = 5 * 4 * 3 * 2!$$

$$5! = 5 * 4 * 3!$$

$$5! = 5 * 4!$$

- **Vejamos o código!**

53

- **Passos para encontrarmos o Big-O da recursividade:**
  - **1. Calcular a complexidade de uma única chamada da função**
  - **2. Expressar o número de chamadas recursivas através dos parâmetros de entrada**
  - **3. Multiplicar o número de chamadas recursivas pela complexidade de uma chamada da recursão**

54

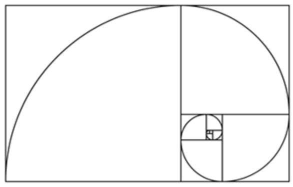
- **1. Complexidade de uma chamada da função: O(1)**
- **2. Quantidade de chamadas recursivas: n**
- **3. Por fim, fazemos:**

$$O(1).n = O(n)$$

55

Exemplo: série de Fibonacci

- **Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89...**



Pyty/Shutterstock

56

- **Vejamos o código!**

57

fib (5)

fib (4) ***fib (3)***

fib (3) ***fib (2)*** ***fib (2)*** ***fib (1)***

fib (2) ***fib (1)***

58

Equação geral da recursividade

$$O(uma\ chamada) * O\left(n^{o}\ de\ chamadas^{n^{o}\ níveis}\right)$$

- **Restrições:**
  - **Número de chamadas deve ser estritamente maior do que 1**
  - **Se o número de chamadas for igual a 1, teremos uma cadeia linear de chamadas (como na fatorial) (...)**

59

- **(...) Se a recursão ocorre em um loop, o número de chamadas depende de cada caso específico. Veremos isso melhor depois**

60

$O(uma\ chamada) * O(n.\ de\ chamadas^{n.\ níveis})$

$O(uma\ chamada) = O(1)$

$O(n.\ de\ chamadas^{n.\ níveis}) = O(2^n)$

$O(1).\ O(2^n) = O(2^n)$

61

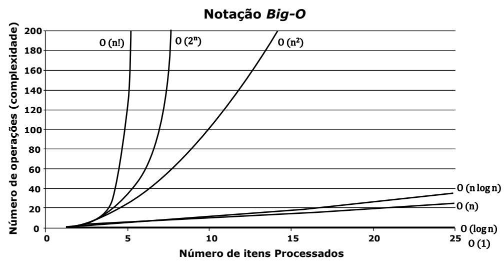
Finalizando

62

Resumo

- **Algoritmo sem iterações nem recursão: O(1)**
- **Laço iterativo simples: O(n)**
- **Progressão aritmética (PA): O(n²)**
- **Progressão geométrica (PG):** $O(2^n)$
- **Dividir para conquistar:** $O(logn)$
- **Recursão simples: O(n)**
- **Recursão em árvore binária:** $O(2^n)$

63



64

Referências

65

- **DROZDEK, A. Estrutura de dados e algoritmos em C++. Trad. da 4. ed. norte-americana. São Paulo: Cengage Learning Brasil, 2018.**
- **KOFFMAN, E. B.; WOLFGANG, P. A. T. Objetos, abstração, estrutura de dados e projeto usando C++. São Paulo: Grupo GEN, 2008.**

66