Revisão de Algoritmos

Prof. Edson Pedro Ferlin

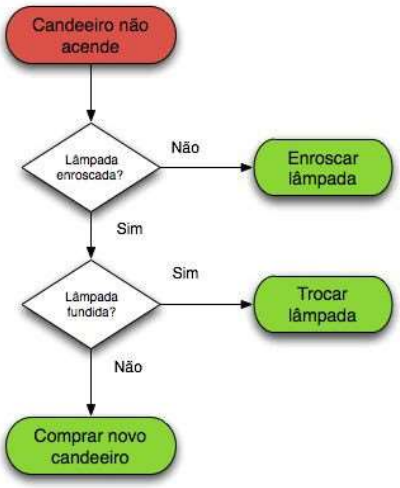
AULA 1 - PROGRAMAÇÃO

Revisão de Algoritmos

```
def revisar(topico):
    while not dominado:
        estudar(topico)
        praticar()
    return "aprovado"

for aula in curso:
    revisar(aula)
```

Conceito



Algoritmo: Procedimento computacional bem definido e finito que, a partir de um valor (ENTRADA) produz uma resposta como saída.

Exemplo de problema computacional

- Problema da ordenação de uma seqüência de números
  - Entrada: seqüência de n números a1, ..., an
  - Saída: permutação da entrada tal que a1 ≤ ... ≤ an
  - Exemplo:
    - Entrada: 31, 41, 59, 26, 41, 58
    - Saída: 26, 31, 41, 41, 58, 59
- Existem vários algoritmos corretos para solucionar esse problema
  - Bubble Sort, Quick Sort, Heap Sort, etc

Se existem vários algoritmos corretos, como saber qual é o melhor?

- 1ª Idéia – Comparar o tempo de execução
  - Não é válida uma vez que parâmetros como o desempenho da máquina e até mesmo a técnica de programação interferem nos resultados.
- 2a Idéia – Mapear cada algoritmo em uma função matemática que "represente" o desempenho do algoritmo
  - Essa idéia foi adotada pela literatura e o nome dessa função é complexidade
  - Para representar a complexidade de um algoritmo usamos a notação de Big O

Complexidade de Algoritmos

- A Complexidade de um algoritmo é a medida da "quantidade de trabalho" para resolver um problema
  - Não pode ser descrita por um número
- Normalmente a quantidade de operações elementares necessárias para resolver o problema depende da instância
  - A complexidade assintótica é definida pelo crescimento da complexidade para entradas suficientemente grandes.

Complexidade Pessimista

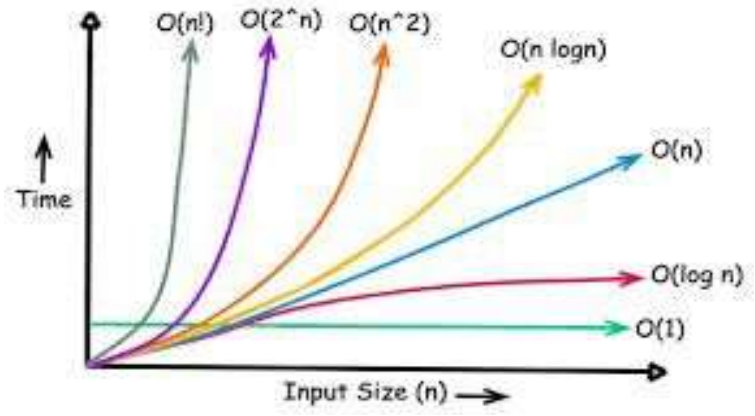
- O critério de avaliação da complexidade do pior caso é o mais utilizado.
- Para essa avaliação os critérios são dependentes somente da estrutura do algoritmo.
- Serão analisadas as operações:
  - Atribuição
  - Seqüência
  - Condicional
  - Iteração definida
  - Iteração indefinida

Notação de Big O

- A notação Big O é um conceito matemático usado na computação para classificar algoritmos pela sua eficiência (tempo de execução ou uso de memória) à medida que os dados de entrada (n) crescem
  - A notação O define uma cota assintótica superior
  - A complexidade de tempo é a mais comum

- Exemplos
  - O(1) - O desempenho do algoritmo é constante independentemente do tamanho da entrada
  - O(n) - O desempenho do algoritmo é diretamente proporcional ao tamanho da entrada
  - O(n²) - O desempenho do algoritmo varia com o quadrado da entrada

Desempenho de Algoritmos (Big O)



Revisão de Algoritmos

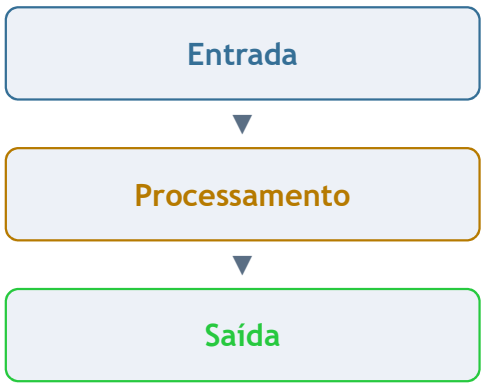
O que é um algoritmo?

DEFINIÇÃO

Uma sequência finita e ordenada de passos que resolve um problema ou executa uma tarefa.

CARACTERÍSTICAS

Entrada	Processamento
Saída	Precisão
Sequência lógica	Finitude



Algoritmos no cotidiano

01	02	03	04	05
Receita culinária	GPS	Caixa eletrônico	Máquina de café	Semáforo inteligente

Objetivo: mostrar que os algoritmos existem muito antes da programação — eles orientam decisões e tarefas do dia a dia.

Como representar algoritmos

1	**Linguagem Natural**	Descrição em texto comum, do jeito que falamos.
2	**Fluxograma**	Representação visual com símbolos e setas.
3	**Portugol**	Pseudolinguagem em português, próxima ao código.
4	**Pseudocódigo**	Passos estruturados, independentes de linguagem.
5	**Linguagens de Programação**	Código executável — ex.: Python.

Declaração de Variáveis

- Uma variável é um espaço na memória onde armazenamos valores;
- Em Python, não é necessário declarar o tipo explicitamente;
- O nome deve começar com uma letra ou _, mas não com números;
- Sensível a maiúsculas e minúsculas (idade ≠ Idade);
- Para declarar uma variável x com valor 10, escrevemos simplesmente o código x=10.

Operações Básicas

Operação	Símbolo	Exemplo	Resultado
Adição	`+`	`10 + 5`	`15`
Subtração	`-`	`10 - 5`	`5`
Multiplicação	`*`	`10 * 5`	`50`
Divisão	`/`	`10 / 3`	`3.3333`
Divisão inteira	`//`	`10 // 3`	`3`
Módulo (resto)	`%`	`10 % 3`	`1`
Exponenciação	`**`	`2 ** 3`	`8`

Estruturas Condicionais

- Podemos utilizar os operadores **if**, **else** e **elif**
- if é utilizado para testar uma condição
- elif é utilizado para se testar uma nova condição caso a condição if anterior seja falsa
- else é utilizado para executar o código se a condição if anterior for falsa

```
idade = 16
if idade >= 18:
  print('Maior de idade')
elif idade >= 14:
  print('Adolescente')
else:
  print('Criança')
```

```
Adolescente
```


Laços de Repetição

- O loop **while** é uma estrutura de repetição que executa um bloco de código enquanto uma condição for verdadeira
- O loop **for** é usado para percorrer elementos de uma sequência, como listas, tuplas, strings, dicionários e até intervalos numéricos
- **continue** pula uma iteração e segue para a próxima
- **break** para o loop antes que ele termine naturalmente

```
for numero in range(1, 10):
    if numero == 5:
        print("Número 5 encontrado! Parando o loop...")
        break  # Sai do loop imediatamente
    print(f"Número: {numero}")
```

```
Número: 1
Número: 2
Número: 3
Número: 4
Número 5 encontrado! Parando o loop...
```

Recursividade

- Característica de um procedimento que contém pelo menos uma chamada a ele mesmo
- Útil para resolver problemas difíceis
- Exemplo de Algoritmos corretos do fatorial:

```python
def fatorial(n):
    fat = 1
    for i in range(n):
        fat = fat * (i+1)
    return fat
```

```python
def fatorialR(n):
    if n > 1:
        return n*fatorialR(n-1)
    else:
        return 1
```


Variáveis

variaveis.py

```python
nome = "Maria"
idade = 20
altura = 1.68
aprovado = True

print(nome)
print(idade)
print(altura)
print(aprovado)
```

CONCEITO

Variáveis armazenam informações na memória durante a execução do programa.

SAÍDA

```
Maria
20
1.68
True
```

Tipos de Dados

tipos.py

```
inteiro = 10          # int
real = 3.14           # float
texto = "Python"      # str
letra = 'A'           # str
logico = False        # bool

print(type(inteiro))
print(type(real))
print(type(texto))
print(type(logico))
```

type()

- int — inteiros
- float — reais
- str — texto
- bool — True/False

Entrada de Dados

entrada.py

```
nome = input("Digite seu nome: ")
idade = int(input("Digite sua idade: "))

print("Nome:", nome)
print("Idade:", idade)
```

input()

input() sempre lê texto. Use int() ou float() para converter o valor digitado em número.

Operadores Aritméticos

aritmeticos.py

```
a = 12
b = 5

print(a + b)
print(a - b)
print(a * b)
print(a / b)
print(a % b)
print(a ** b)
```

Aritméticos

- + soma
- - subtração
- * multiplicação
- / divisão
- % resto
- ** potência

Operadores Relacionais

relacionais.py

```
idade = 18

print(idade > 16)
print(idade >= 18)
print(idade == 20)
print(idade != 15)
```

Comparação

- > maior
- >= maior ou igual
- == igual
- != diferente

Operadores Lógicos

logicos.py

```
idade = 20
possui_carteira = True

if idade >= 18 and possui_carteira:
    print("Pode dirigir")
```

and / or / not

Combinam condições. Com and, todas precisam ser verdadeiras para o bloco executar.

Estrutura IF

if.py

```
nota = float(input("Nota: "))

if nota >= 6:
    print("Aprovado")
else:
    print("Reprovado")
```

Decisão

Executa um bloco quando a condição é verdadeira; caso contrário, executa o else.

IF Aninhado

if_elif.py

```python
nota = float(input("Nota: "))

if nota >= 9:
    print("Excelente")
elif nota >= 7:
    print("Bom")
elif nota >= 6:
    print("Regular")
else:
    print("Insuficiente")
```

elif

Encadeia várias condições em sequência, testadas de cima para baixo até uma ser verdadeira.

Match (Python 3.10+)

match.py

```python
opcao = int(input("Escolha: "))

match opcao:
    case 1:
        print("Novo")
    case 2:
        print("Abrir")
    case 3:
        print("Salvar")
    case _:
        print("Inválido")
```

match / case

Compara um valor com vários casos. O case _ funciona como padrão (qualquer outro valor).

Atividade — Número positivo

exercicio.py

```
numero = int(input("Número: "))

if numero > 0:
    print("Positivo")
elif numero < 0:
    print("Negativo")
else:
    print("Zero")
```

Objetivo

Classifique o número informado como positivo, negativo ou zero.

Laço FOR

for.py

```
for i in range(1, 11):
    print(i)
```

range(início, fim)

Repete o bloco para cada valor da sequência. range(1, 11) gera de 1 a 10.

Tabuada

tabuada.py

```
numero = int(input("Número: "))

for i in range(1, 11):
    print(f"{numero} x {i} = {numero*i}")
```

f-string

O prefixo f permite inserir variáveis diretamente no texto, dentro de chaves { }.

Laço While

while.py

```
senha = ""

while senha != "123":
    senha = input("Senha: ")

print("Acesso liberado")
```

Repetição

Repete o bloco enquanto a condição for verdadeira. Ideal quando não se sabe o número de repetições.

While com Contador

contador.py

```
contador = 1

while contador <= 5:
    print(contador)
    contador += 1
```

Contador

Use uma variável contadora e incremente com += 1 para controlar o número de repetições.

Break e Continue

break — interrompe o laço

break.py

```
for i in range(10):
    if i == 5:
        break
    print(i)
```

continue — pula para a próxima

continue.py

```
for i in range(10):
    if i % 2 == 0:
        continue
    print(i)
```

break encerra o laço imediatamente. **continue** ignora o resto da volta e segue para a próxima iteração.

Atividade — Somar de 1 até 100

soma.py

```
soma = 0

for i in range(1, 101):
    soma += i

print(soma)
```

ATIVIDADE

Acumulador

Some cada valor em uma variável acumuladora. Resultado esperado: 5050.

Funções

funcoes.py

```
def soma(a, b):
    return a + b

print(soma(5, 8))
```

def

Agrupa um trecho de código reutilizável. return devolve o resultado para quem chamou a função.

Recursividade

fatorial.py

```python
def fatorial(n):
    if n == 0:
        return 1
    return n * fatorial(n - 1)

print(fatorial(5))
```

CONCEITO

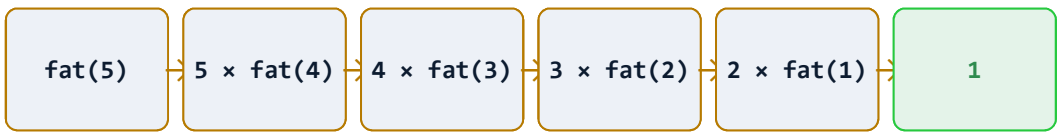
Uma função recursiva chama a si mesma até chegar a um caso-base que encerra a repetição.

SAÍDA

120

Árvore de Recursão

Como fatorial(5) se desdobra em chamadas até o caso-base:



Descida: cada chamada aguarda a próxima. **Retorno:** a partir do caso-base (1), os resultados voltam multiplicando: 1 → 2 → 6 → 24 → 120.

Fibonacci Recursivo

fib_rec.py

```python
def fibonacci(n):
    if n <= 1:
        return n
    return fibonacci(n-1) + fibonacci(n-2)

for i in range(10):
    print(fibonacci(i))
```

COMPLEXIDADE

O(2ⁿ)

Por que é lento?

Recalcula os mesmos valores várias vezes. O custo cresce exponencialmente.

Fibonacci Iterativo

fib_iter.py

```python
a = 0
b = 1

for i in range(10):
    print(a)
    a, b = b, a + b
```

COMPLEXIDADE

O(n)

Bem mais rápido

A versão iterativa calcula cada valor uma única vez. Mesmo resultado, desempenho muito melhor.

Medindo Tempo

tempo.py

```python
import time

inicio = time.time()
soma = 0
for i in range(1000000):
    soma += i
fim = time.time()

print(fim - inicio)
```

Eficiência

O módulo time mede quanto um trecho demora. É o primeiro passo para comparar a eficiência de algoritmos.

Busca Linear

busca_linear.py

```python
vetor = [12, 7, 18, 4, 20, 15]
valor = 20

for i in vetor:
    if i == valor:
        print("Encontrado")
```

COMPLEXIDADE

O(n)

Busca Linear

Percorre elemento por elemento. No pior caso, verifica toda a lista.

Busca Binária

busca_binaria.py

```python
import bisect

vetor = [2, 4, 6, 8, 10, 12, 14]
indice = bisect.bisect_left(vetor, 10)

print(indice)
```

COMPLEXIDADE

O(log n)

Busca Binária

Divide a lista ao meio a cada passo. Exige que o vetor esteja ordenado.

Bubble Sort

bubble_sort.py

```python
vetor = [8, 5, 2, 9, 1]

for i in range(len(vetor)):
    for j in range(len(vetor)-1):
        if vetor[j] > vetor[j+1]:
            vetor[j], vetor[j+1] = vetor[j+1], vetor[j]

print(vetor)
```

COMPLEXIDADE

O(n²)

Bubble Sort

Compara pares vizinhos e troca de lugar. Dois laços aninhados tornam-no lento.

Função sorted()

sorted.py

```
vetor = [8, 5, 2, 9, 1]

print(sorted(vetor))
```

COMPLEXIDADE

O(n log n)

Timsort

Internamente o Python usa o Timsort, com complexidade média O(n log n) — muito eficiente.

Comparação de Complexidades

ALGORITMO	COMPLEXIDADE
Acesso a lista	O(1)
Busca Binária	O(log n)
Timsort (sorted)	O(n log n)
Busca Linear	O(n)
Bubble Sort	$O(n^2)$
Fibonacci Recursivo	$O(2^n)$

Atividade - Algoritmos

Faça um programa que leia 10 números e, a partir deles, calcule os resultados abaixo.

01	02	03	04
Ler	**Maior**	**Menor**	**Média**
10 números	encontrar o maior valor	encontrar o menor valor	calcular a média dos valores

Exercícios

1) Escreva uma função curta em Python que receba um inteiro positivo n e retorne a soma dos quadrados de todos os inteiros positivos menores que n.

2) Escreva uma função curta em Python que receba um inteiro positivo n e retorne a soma dos quadrados de todos os inteiros positivos ímpares menores que n.

3) Escreva uma função curta em Python minmax(data) que receba uma sequência de um ou mais números e retorne o menor e o maior valor na forma de uma tupla de tamanho dois. Não utilize as funções embutidas min() ou max().

4) Escreva uma função curta em Python is_multiple(n, m) que receba dois valores inteiros e retorne True se n for múltiplo de m, ou seja, se existir um inteiro i tal que n = m * i, e False caso contrário.

5) Quais parâmetros devem ser passados ao construtor range() para produzir a sequência: 50, 60, 70, 80?

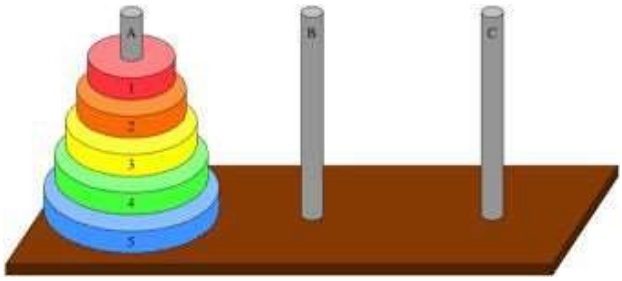
6) Quais parâmetros devem ser passados ao construtor range() para produzir a sequência: 8, 6, 4, 2, 0, -2, -4, -6, -8?

7) O Python permite o uso de índices negativos em sequências (como strings). Se uma string s tem comprimento n e usamos s[k] com $-n \leq k < 0$, qual é o índice equivalente $j \geq 0$ tal que s[j] referência o mesmo elemento?

8) Solicite uma palavra ao usuário e verifique se ela é um palíndromo (lida de trás para frente, ela continua igual). Dica: Pense de forma recursiva.

Problema

Problema das Torres de Hanoi – São dados 3 pinos A,B e C, n discos de tamanho diferentes, empilhados em ordem decrescente de diâmetro no pino A. Mover a pilha do pino A para o pino B, um disco de cada vez, sendo que é proibido empilhar um disco maior em cima de um menor.



Contato

eferlin@live.com

(BLOG) professorferlin.blogspot.com

(SITE) professorferlin.com.br

(YOUTUBE) ProfEdsonPedroFerlin