


Análise de Algoritmos



Professor: José Carlos Althoff Departamento de COMPUTAÇÃO / MATEMÁTICA
Universidade Regional de Blumenau FURB

O que Você
entende por análise
de Algoritmos?

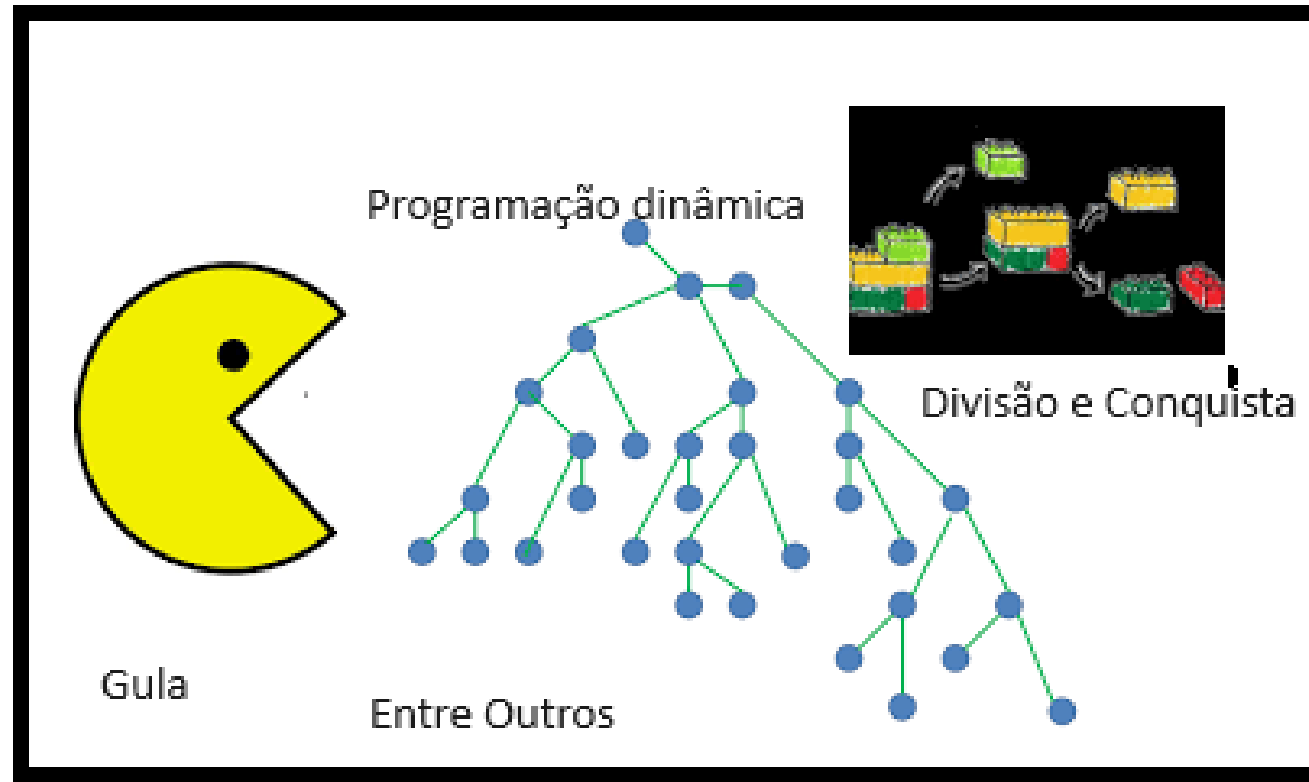
Análise de algoritmos

estuda
a correção  e o desempenho
de algoritmos.

Em outras palavras, a análise de algoritmos procura respostas para perguntas do seguinte tipo:

- ▶ Esse Algoritmo resolve o meu problema?
- ▶ Quanto tempo esse algoritmo consome para processar uma entrada de tamanho “n”?

Além disso, a análise de algoritmos estuda certos paradigmas como:



Os quais se mostraram úteis na criação de algoritmos para diversos **problemas** computacionais.

O que é um



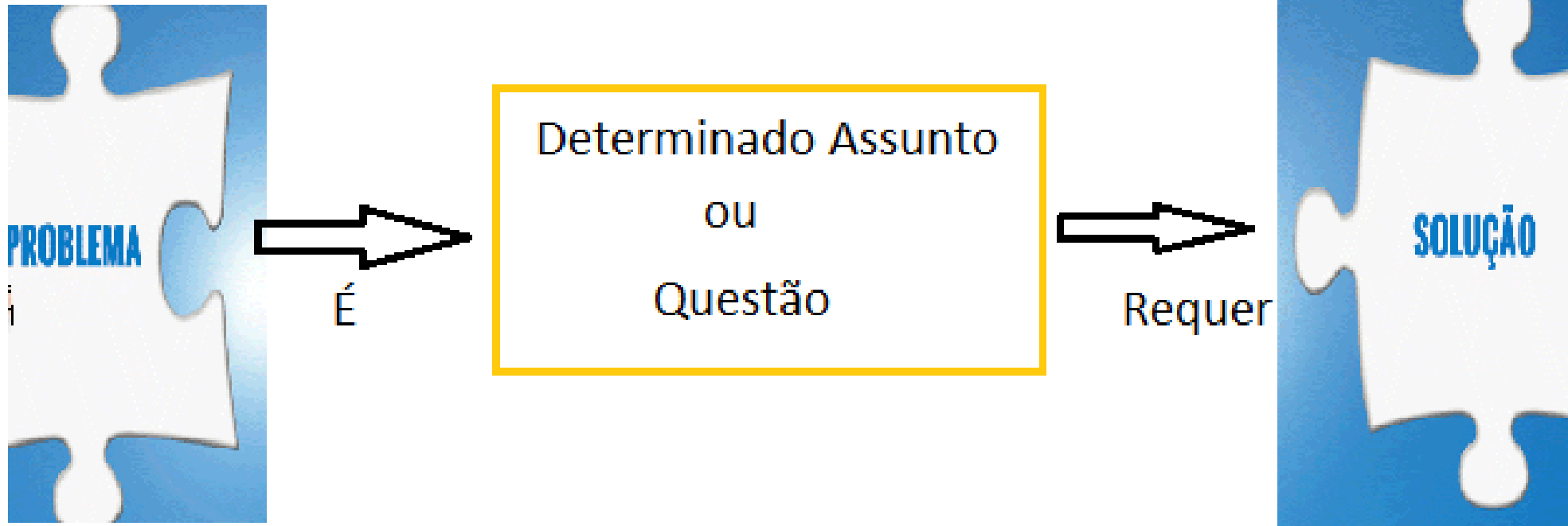
?



- Na acepção científica, “problema é qualquer questão não resolvida e que é objeto de discussão, em qualquer domínio do conhecimento” (GIL, 1999, p.49).

•Problema, para Kerlinger (1980, p.35), “é uma questão que mostra uma situação necessitada de discussão, investigação, decisão ou solução”.

Resumindo



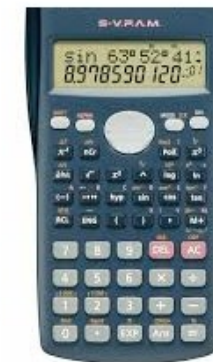
É importante mencionar que, na grande maioria dos casos, um problema é algo difícil de ser solucionado.

Porém não é impossível, mesmo que isso seja difícil para um ser humano.



Um exemplo Matemático:

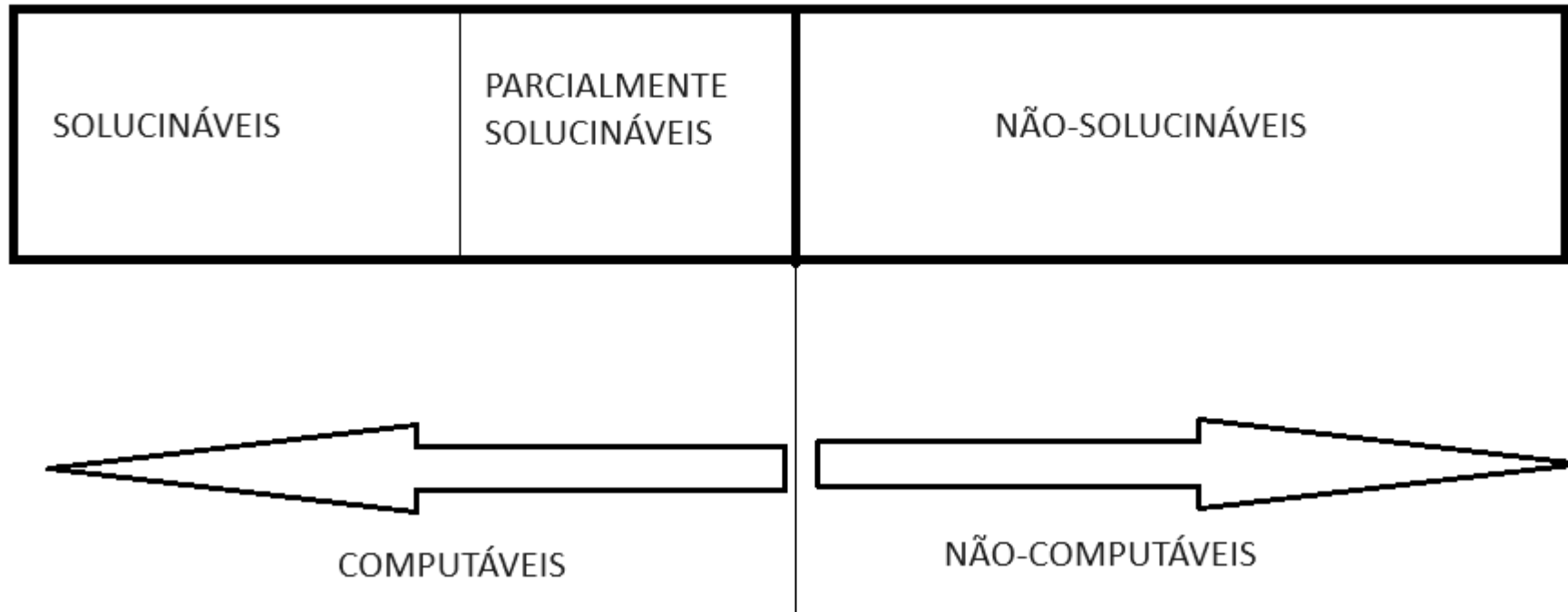
Existem cálculos com vários dígitos, que para um ser humano comum, seria praticamente impossível de conseguir resolver, mas mediante á um aparelho, como uma calculadora, pode ser resolvido em questão de segundos.



Já na Computação temos:

- ▶ O objetivo do estudo de problemas é investigar a existência ou não de algoritmos que solucionem determinada classe de problemas.
- ▶ Ou seja, investigar os limites da compatibilidade e, conseqüentemente, os do que pode efetivamente ser implementado em um computador.
- ▶ Em particular, o estudo da solucionabilidade objetiva evitar a pesquisa de soluções inexistentes.

NO UNIVERSO DE TODOS OS PROBLEMAS



Aqui vamos nos ater a problemas que sejam possíveis de serem tratados por algoritmos.

Exemplos:

- 1 - A ordenação de um conjunto de entrada;
- 2 - Como achar uma melhor rota de entrega de um determinado produto;
- 3 - Como proceder o corte de uma placa de metal da melhor maneira, para haver o menor desperdício possível do material, etc.

Problemas e suas instâncias.

Todo problema computacional é uma coleção de casos particulares que chamaremos *instâncias*.

Uma instância é especificada quando atribuímos valores aos parâmetros do problema.

Em outras palavras, uma instância é especificada por um particular conjunto de dados do problema.

Observação

A palavra **instância** é um neologismo, importado do inglês.

Ela está sendo empregada aqui no sentido de *exemplo, exemplar, espécime, amostra, ilustração*.

Veja alguns exemplos:

1) Problema da multiplicação de números naturais: Dados números **naturais** u e v , encontrar a expansão decimal do produto $u \times v$.

Cada instância do problema é definida por dois números naturais. Por exemplo, os números 3141 e 141421 definem uma instância.

2) Problema da equação inteira do segundo grau: Dados números **inteiros** a , b e c , encontrar um número inteiro x tal que $ax^2 + bx + c = 0$.

Cada instância do problema é definida pelos valores de a , b e c . Por exemplo, os números 472, -311 e 57281 definem uma instância do problema; essa instância consiste em encontrar um número inteiro x tal que $472x^2 - 311x + 57281 = 0$.

3) Problema da ordenação de vetor inteiro: Rearranjar (ou seja, permutar) os elementos de um vetor $A[1..n]$ de números inteiros de modo que ele fique crescente.

Cada instância do problema é definida por um número natural n e um vetor $A[1..n]$ de inteiros. Por exemplo, o número 5 e o vetor (876, 145, 323, 112, 221) definem uma instância do problema; essa instância consiste em rearranjar o vetor (876, 145, 323, 112, 221) em ordem crescente.

Em geral, **nem toda** instância de um problema tem solução. Assim, por exemplo, a instância $(1, 2, 3)$ do problema da equação do segundo grau não tem solução, pois não existe um número **inteiro** x tal que $1x^2 + 2x + 3 = 0$.

A solução de uma instância de um dado problema pode ser um número, um vetor, um valor booleano, etc., dependendo da natureza do problema. Já a solução de um *problema* é sempre um *algoritmo*.

Dizemos que um algoritmo *resolve* um dado problema se, ao receber a descrição de *qualquer instância* do problema, devolve uma solução da instância ou informa que a instância não tem solução.

Essa exigência é deveras pesada, pois obriga o algoritmo a resolver não só as instâncias que aparecem em aplicações práticas como também aquelas instâncias "patológicas" que nem parecem razoáveis.

OBS: Patológicas no sentido de alterações não razoáveis.

Exercícios

1. Dê uma solução da instância $(1, 2, 1)$ do problema da equação inteira do segundo grau.
2. Dê uma solução da instância $(1, \frac{3}{4}, 0)$ do problema da equação inteira do segundo grau.
3. Dê uma solução da instância $(4, -2, 8, 6)$ do problema da ordenação de vetor inteiro.

Então,

Algoritmos resolvem problemas.

► Algoritmos contribuíram para a solução de problemas, da sociedade, e conseqüentemente para evolução tecnológica vista nas últimas décadas e são cada vez mais complexos.





► Quando se fala em algoritmo, muitas pessoas pensam rapidamente em computadores, tecnologia e até mesmo códigos difíceis de serem compreendidos.

No entanto, o conceito e a aplicação são bem mais simples do que parecem.



Os algoritmos datam de tempos babilônicos, mas tornaram-se mais conhecidos na modernidade, principalmente, quando associados aos computadores e às estratégias de otimização para buscadores.



Um **algoritmo** é uma sequência de instruções ou comandos realizados de maneira sistemática com o objetivo de resolver um **problema** ou executar uma tarefa.

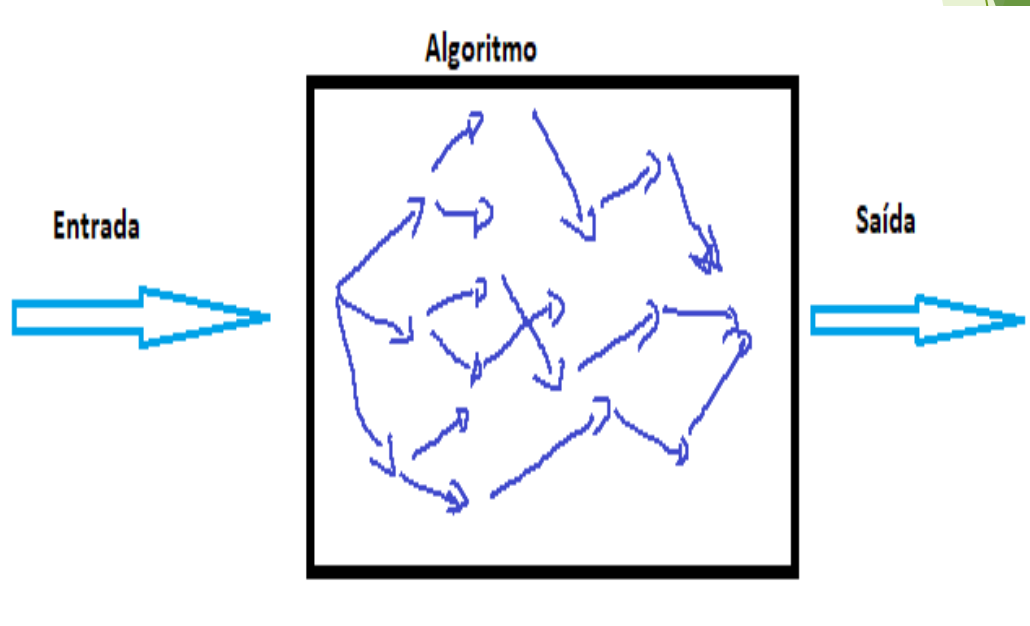
Todas as funções dos computadores, smartphones e tablets, por **exemplo**, são resultado de **algoritmos**.

Algoritmo a partir da visão
computacional.

Algoritmo é qualquer procedimento computacional bem definido que torna algum valor ou conjunto de valores como entrada e produz algum valor ou conjunto de valores como saída.

Resumindo:

Um algoritmo é uma sequência de etapas computacionais que transforma a entrada na saída.



É fundamental compreender que o algoritmo se justifica no resultado que ele almeja alcançar, logo, deve ter um objetivo específico.

Uma sequência de instruções simples pode se tornar mais complexa conforme a necessidade de considerar outras situações.

Segundo Edsger Dijkstra (Cientista da computação holandês - Área de algoritmos)

“UM ALGORITMO CORRESPONDE A UMA DESCRIÇÃO DE UM PADRÃO DE COMPORTAMENTO, EXPRESSO EM TERMOS DE UM CONJUNTO FINITO DE AÇÕES”.

Exercício 1

- Criar um algoritmo que resolva o problema abaixo:

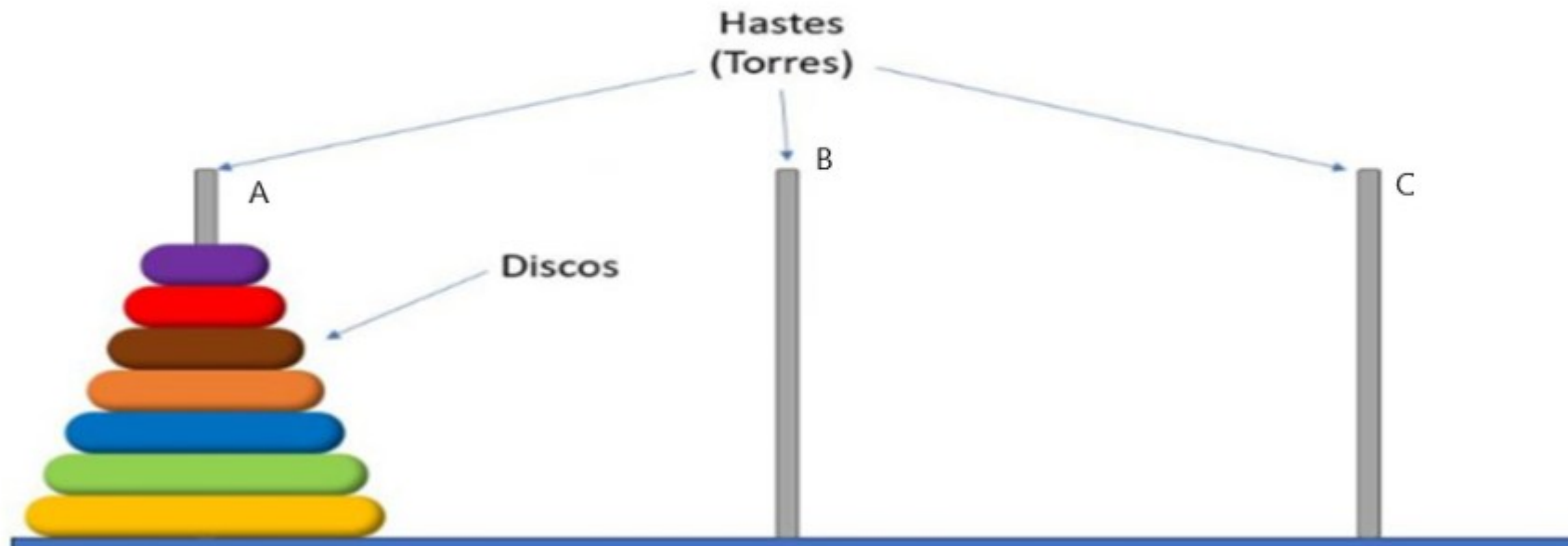
Imagine uma torneira com defeito a qual enche um copo de 180 ml em 12 minutos e 30 segundos.

- 1) Determine a quantidade de água que esta torneira com defeito deixa escapar em 24 horas em litros.
- 2) Este vazamento em relação a uma casa é significativo em 30 dias?
- 3) Apresente sua solução.

Exercício 2

Construir um algoritmo em Python que resolva o problema abaixo para 5 discos.

A **Torre de Hanoi** é um famoso quebra-cabeça ou problema matemático, inventado pelo matemático francês François Édouard Anatole Lucas no século XIX, que consiste em mover uma pilha de discos de tamanhos diferentes de uma haste de origem para uma haste de destino, usando uma haste intermediária. A regra básica é que um disco maior nunca pode ficar em cima de um disco menor.



Exemplo de resposta.

```
Mova o disco 1 de A para C  
Mova o disco 2 de A para B  
Mova o disco 1 de C para B  
Mova o disco 3 de A para C  
Mova o disco 1 de B para A  
Mova o disco 2 de B para C  
Mova o disco 1 de A para C
```



```
def hanoi(n, origem, destino, auxiliar):  
    if n == 1:  
        print(f"Mover disco 1 de {origem} para {destino}")  
        return  
    hanoi(n-1, origem, auxiliar, destino)  
    print(f"Mover disco {n} de {origem} para {destino}")  
    hanoi(n-1, auxiliar, destino, origem)  
  
# Chamando a função para resolver a Torre de Hanói com 5 discos  
hanoi(5, 'A', 'C', 'B')
```


Mover disco 1 de A para C
Mover disco 2 de A para B
Mover disco 1 de C para B
Mover disco 3 de A para C
Mover disco 1 de B para A
Mover disco 2 de B para C
Mover disco 1 de A para C
Mover disco 4 de A para B
Mover disco 1 de C para B
Mover disco 2 de C para A
Mover disco 1 de B para A
Mover disco 3 de C para B
Mover disco 1 de A para C
Mover disco 2 de A para B
Mover disco 1 de C para B
Mover disco 5 de A para C
Mover disco 1 de B para A
Mover disco 2 de B para C
Mover disco 1 de A para C
Mover disco 3 de B para A
Mover disco 1 de C para B
Mover disco 2 de C para A
Mover disco 1 de B para A
Mover disco 4 de B para C
Mover disco 1 de A para C
Mover disco 2 de A para B

Mover disco 1 de C para B
Mover disco 3 de A para C
Mover disco 1 de B para A
Mover disco 2 de B para C
Mover disco 1 de A para C

⚡ Total de movimentos necessários: $2^5 - 1 = 31$

python

```
def torre_de_hanoi(n, origem, destino, auxiliar):
```

```
    """
```

```
    Função recursiva para resolver a Torre de Hanói.
```

```
    Parâmetros:
```

```
    n          -> Número de discos
```

```
    origem     -> Pino de onde os discos saem
```

```
    destino    -> Pino onde os discos devem ser colocados
```

```
    auxiliar   -> Pino auxiliar para movimentação
```

```
if n == 1: # Caso base: mover apenas um disco diretamente
    print(f"Mova o disco 1 de {origem} para {destino}")
    return
```

```
# 1º Passo: Mover (n-1) discos da origem para o pino auxiliar
torre_de_hanoi(n - 1, origem, auxiliar, destino)
```

```
# 2º Passo: Mover o maior disco restante diretamente para o destino
print(f"Mova o disco {n} de {origem} para {destino}")
```

```
# 3º Passo: Mover os (n-1) discos do pino auxiliar para o destino
torre_de_hanoi(n - 1, auxiliar, destino, origem)
```

Número de discos = 5

```
num_discos = 5
```

Chamando a função com os três pinos: A (origem), C (destino) e B (auxiliar)

```
torre_de_hanoi(num_discos, 'A', 'C', 'B')
```

Referências

- ▶ https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/instance.html

KERLINGER, Fred Nichols. Metodologia da Pesquisa em Ciências Sociais: um tratamento conceitual / Fred N. Kerlinger; [tradução Helena Mendes Rotundo; revisão técnica José Roberto Malufe]. São Paulo: EPU: EDUSP - Editora da Universidade de São Paulo, 1980.