

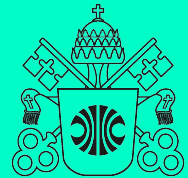
# ANÁLISE TEÓRICA E EXPERIMENTAL DO PROBLEMA DA MOCHILA 0-1



**Autores:** Alberto Magno, Leonardo, Josué

**Professor:** Prof. Walisson Ferreira de Carvalho

**PUC Minas - Junho/2025**



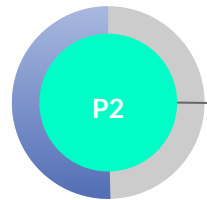
**PUC Minas**

# ROTEIRO



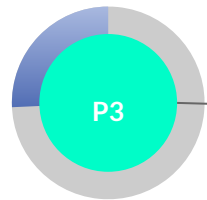
P1

Definição e Prova de NP-Compleitude



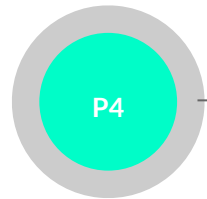
P2

Importância e Aplicações



P3

Algoritmos, Implementações e Comparações



P4

Análise Experimental dos Resultados



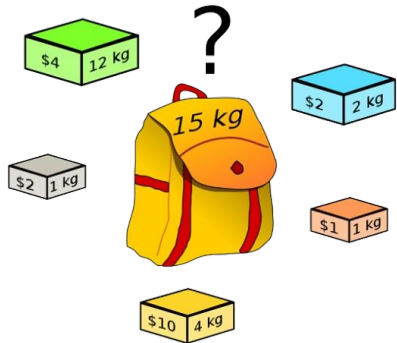


# PARTE 1 – DEFINIÇÃO E PROVA

# INTRODUÇÃO AO PROBLEMA DA MOCHILA 0-1

- EXPLICAÇÃO DO PROBLEMA:

- ESCOLHA DOS MELHORES ITENS DENTRO DE UMA MOCHILA COM PESO LIMITADO.



- DEFINIÇÃO MATEMÁTICA FORMAL:

$$\text{MAXIMIZAR: } \sum_{i=1}^n v_i x_i \quad \text{SUJEITO A: } \sum_{i=1}^n w_i x_i \leq W$$

Onde:

- $n$ : é o número total de itens disponíveis.
- $i$ : representa o índice de um item específico, de 1 a  $n$ .
- $v_i$ : é o valor do item  $i$ .
- $w_i$ : é o peso do item  $i$ .
- $W$ : é a capacidade máxima de peso da mochila.
- $x_i \in \{0, 1\}$ : é a variável de decisão.  $x_i = 1$  se o item  $i$  for escolhido, e  $x_i = 0$  caso contrário.

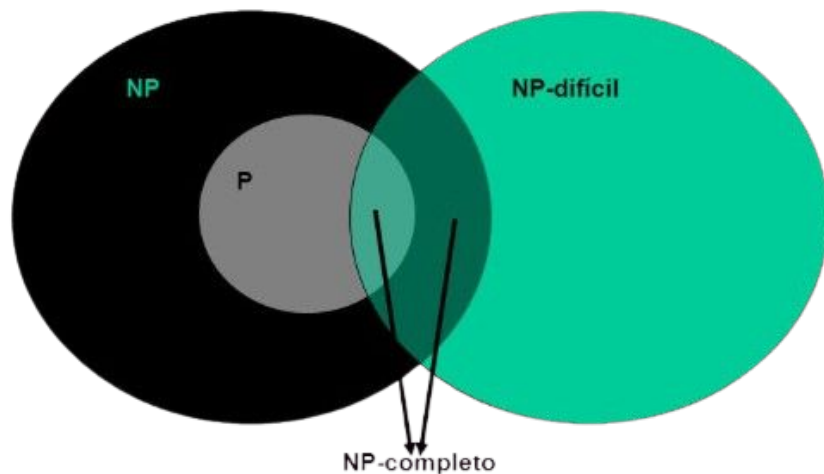
# ANALOGIAS



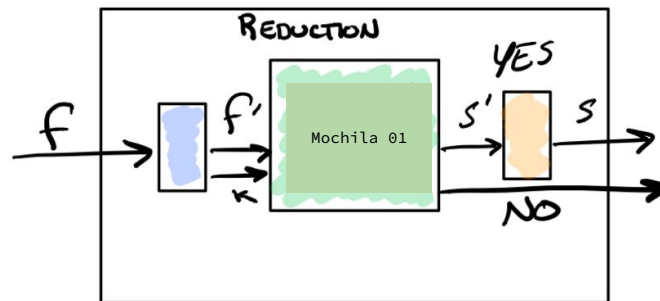
- A ANALOGIA CLÁSSICA: IMAGINE UM VIAJANTE QUE PRECISA ARRUMAR SUA MOCHILA COM UM CONJUNTO DE ITENS.
- O OBJETIVO: O OBJETIVO É ESCOLHER QUAIS ITENS LEVAR PARA MAXIMIZAR O VALOR TOTAL DE SUA BAGAGEM.
- A RESTRIÇÃO PRINCIPAL: CAPACIDADE MÁXIMA DE PESO QUE A MOCHILA PODE SUPORTAR.
- A REGRA "0-1": PARA CADA ITEM, A DECISÃO É BINÁRIA
  - NÃO É POSSÍVEL LEVAR FRAÇÕES DE UM ITEM!

# PROVA DE QUE É NP-COMPLETO

- VERIFICAÇÃO RÁPIDA (PERTENCE A NP)
  - DADA UMA POSSÍVEL SOLUÇÃO, SUA VALIDADE É VERIFICÁVEL EM TEMPO POLINOMIAL.
- PROBLEMA CONHECIDO (É NP-DIFÍCIL)
  - REDUÇÃO POLINOMIAL A PARTIR DO PROBLEMA DA SOMA DE SUBCONJUNTOS (SUBSET SUM).
- CONCLUSÃO:
  - UM PROBLEMA EM NP QUE TAMBÉM É NP-DIFÍCIL É, POR DEFINIÇÃO, NP-COMPLETO.



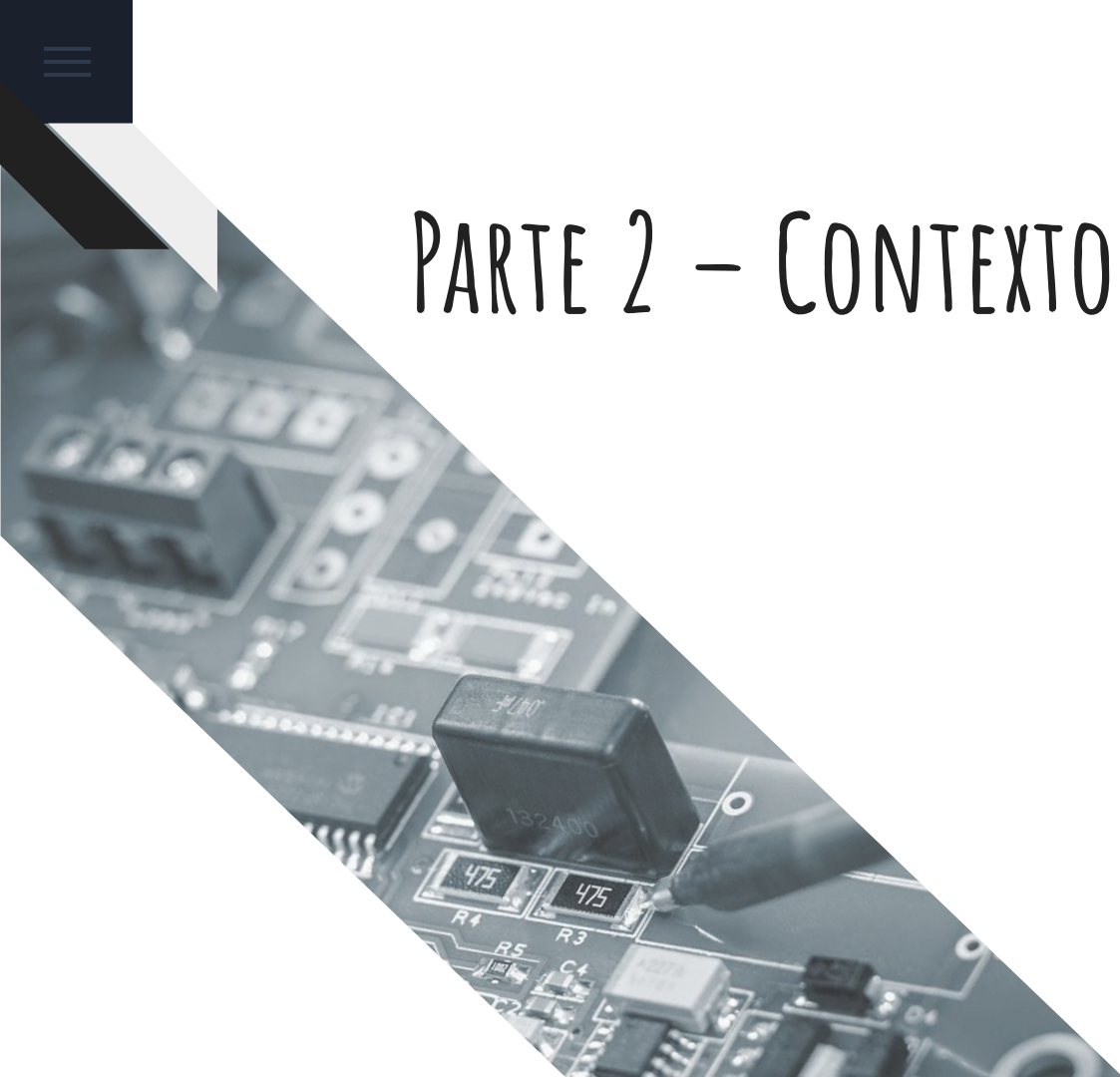
# REDUÇÃO SUBSET SUM



1. PROBLEMA DE PARTIDA ( $F$ ): (SUBSET SUM).
  - INPUT: UM CONJUNTO DE INTEIROS  $S$  E UM NÚMERO ALVO  $T$ .
  - PERGUNTA: EXISTE UM SUBCONJUNTO DE  $S$  CUJA SOMA É EXATAMENTE  $T$ ?
2. A TRANSFORMAÇÃO ( $F \rightarrow F'$ ):
  - PARA CADA NÚMERO  $s$  ( $i$ ) NO CONJUNTO  $S$ , CRIE UM ITEM  $i$ .
  - DEFINA O PESO E O VALOR DO ITEM  $i$  COMO O PRÓPRIO NÚMERO:  $w(i) \leftarrow s(i) \mid v(i) \leftarrow s(i)$ .
3. NOVO PROBLEMA ( $F'$ ): UMA INSTÂNCIA DO PROBLEMA DA MOCHILA 0-1.
  - DEFINA A CAPACIDADE MÁXIMA DA MOCHILA  $W$  COMO O ALVO  $T$ :  $W \leftarrow T$ .



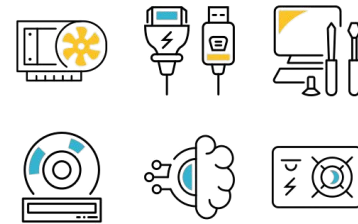
# PARTE 2 – CONTEXTO E APLICAÇÕES





# IMPORTÂNCIA DO PROBLEMA

- MODELO FUNDAMENTAL PARA OTIMIZAÇÃO
  - SERVE COMO BASE PARA RESOLVER PROBLEMAS DE DECISÃO COM RECURSOS LIMITADOS.
- APLICAÇÕES
  - APLICAÇÕES EM FINANÇAS E NEGÓCIOS
  - APLICAÇÕES EM LOGÍSTICA E ENGENHARIA
  - APLICAÇÕES EM COMPUTAÇÃO



# APLICAÇÃO REAL EM LOGÍSTICA

- CENÁRIO: CARREGAMENTO DE VEÍCULOS (CAMINHÕES, CONTÊINERES, AVIÕES) COM CAPACIDADE DE PESO OU VOLUME LIMITADA.
- MAPEAMENTO PARA A MOCHILA:
  - MOCHILA = CAPACIDADE TOTAL DO VEÍCULO (W).
  - ITENS = PACOTES, CAIXAS OU CARGAS A SEREM TRANSPORTADAS.
  - PESO = PESO OU VOLUME DE PACOTE
  - VALOR = FINANCEIRO OU PRIORIDADE DE ENTREGA



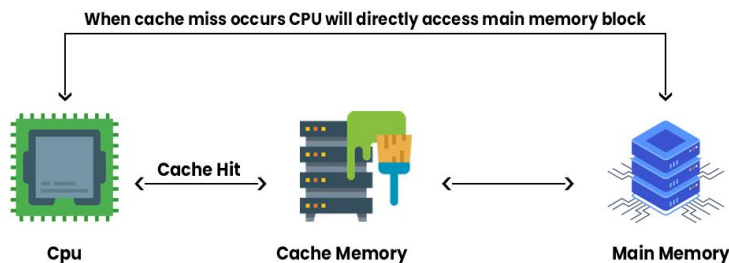
## Aplicações do problema da mochila

Embora possa parecer simples quando pensamos do ponto de vista doméstico, o problema da mochila tem diversas aplicações, como na área da logística e na cadeia de suprimentos.

<https://www.mecalux.com.br/blog/problema-da-mochila>

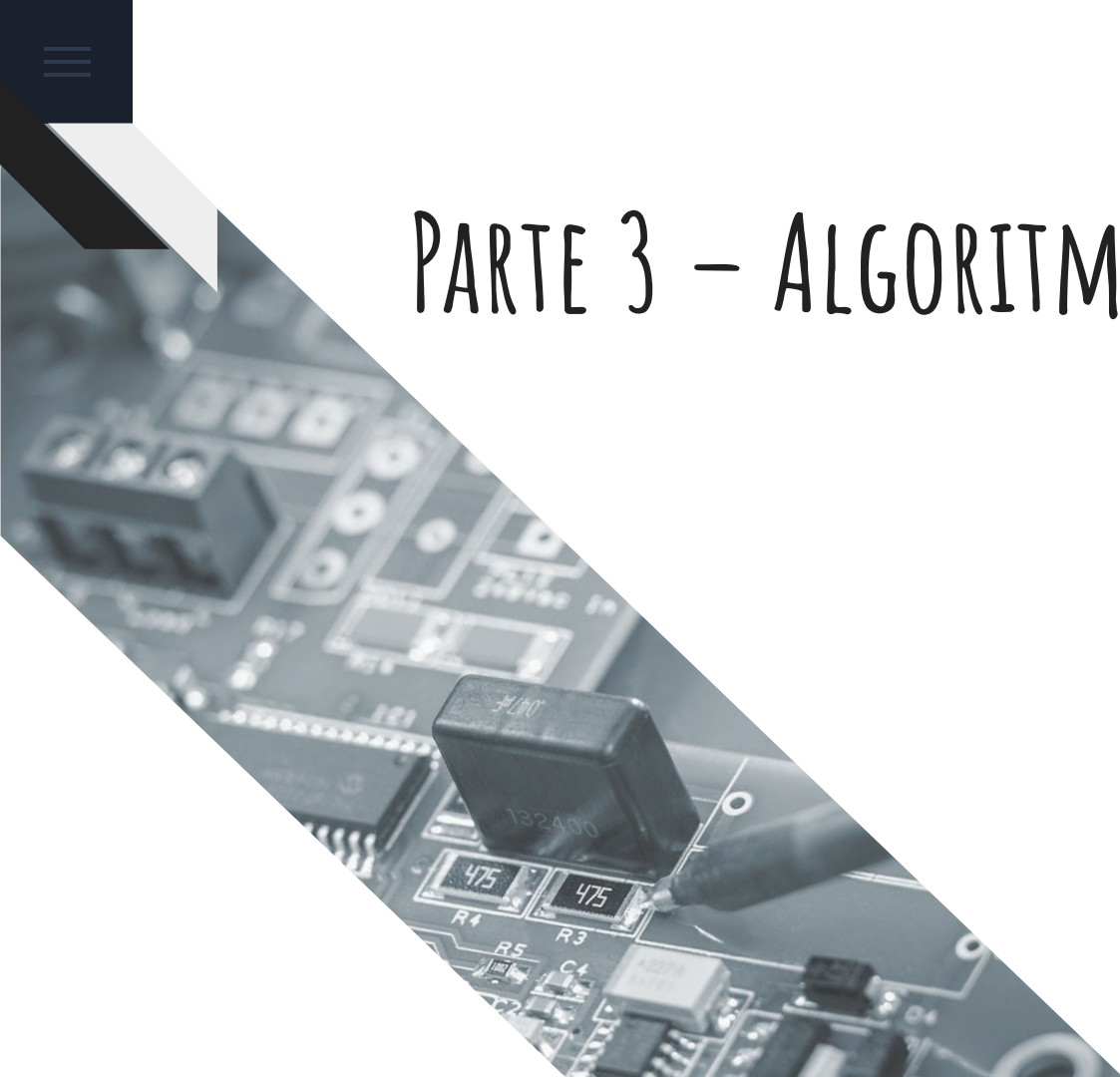
# APLICAÇÃO REAL EM COMPUTAÇÃO

- CENÁRIO: GERENCIAR UMA MEMÓRIA CACHE.
- MAPEAMENTO PARA A MOCHILA:
  - MOCHILA = TAMANHO TOTAL DO CACHE (W).
  - ITENS = DADOS QUE PODEM SER ARMAZENADOS.
  - PESO = TAMANHO DO DADO (EM KB OU MB)
  - VALOR = FREQUÊNCIA DE ACESSO AO DADO (CACHE HIT RATE)





# PARTE 3 – ALGORITMOS E COMPARAÇÃO





# ABORDAGENS DE SOLUÇÃO

- ALGORITMOS EXATOS:

- GARANTEM A SOLUÇÃO ÓTIMA.
- O CUSTO COMPUTACIONAL É ALTO
  - EXPONENCIAL
  - PSEUDO-POLINOMIAL.
- EXEMPLOS:
  - FORÇA BRUTA
  - PROGRAMAÇÃO DINÂMICA.
    - TOPDOWN - BOTTON UP

- ALGORITMOS HEURÍSTICOS:

- BUSCAM UMA SOLUÇÃO "BOA" DE FORMA RÁPIDA.
- SEM GARANTIA DE SOLUÇÃO ÓTIMA.
- CUSTO COMPUTACIONAL BAIXO
  - GERALMENTE POLINOMIAL.
- EXEMPLO:
  - ALGORITMO GULOSO.

# SOLUÇÃO EXATA - FORÇA BRUTA COM DIVISÃO E CONQUISTA

- COMPLEXIDADE:
  - $O(2^N)$

```
ALGORITMO MochilaRecursiva(capacidade, pesos, valores, n)
1. SE n = 0 OU capacidade = 0 ENTÃO
2.     RETORNE 0
3. FIM SE

4. SE pesos[n-1] > capacidade ENTÃO
5.     RETORNE MochilaRecursiva(capacidade, pesos, valores,
n-1)
6. FIM SE

7. SENÃO
8.     opcao_incluir <- valores[n-1] +
MochilaRecursiva(capacidade - pesos[n-1], pesos, valores,
n-1)
9.     opcao_nao_incluir <- MochilaRecursiva(capacidade,
pesos, valores, n-1)
10.    RETORNE MÁXIMO(opcao_incluir, opcao_nao_incluir)
11. FIM SENÃO
FIM ALGORITMO
```

# SOLUÇÃO EXATA - PROGRAMAÇÃO DINÂMICA

- TÉCNICA:

- TABULAÇÃO  
(BOTTOM-UP)

- COMPLEXIDADE:

- $O(N \times W)$

```
ALGORITMO MochilaDinamica(capacidade_total, pesos, valores, n)
1.  Crie uma tabela DP de (n + 1) x (capacidade_total + 1).
2.  Inicialize todos os valores da tabela DP com 0.
3.  PARA i DE 1 ATÉ n FAÇA
4.      PARA c DE 1 ATÉ capacidade_total FAÇA
5.          valor_sem_item <- DP[i-1][c]
6.          valor_com_item <- 0
7.
8.          SE pesos[i-1] <= c ENTÃO
9.              valor_com_item <- valores[i-1] + DP[i-1][c
- pesos[i-1]]
10.         FIM SE
11.
12.         DP[i][c] <- MÁXIMO(valor_com_item,
valor_sem_item)
13.     FIM PARA
14. FIM PARA
15. RETORNE DP[n][capacidade_total]
FIM ALGORITMO
```

# COMPLEXIDADE PSEUDO-POLINOMIAL

- O ALGORITMO COM PROGRAMAÇÃO DINÂMICA CRIA UMA TABELA DE TAMANHO  $N+1$  LINHAS E  $W+1$  COLUNAS, ONDE CADA CÉLULA REPRESENTA A MELHOR SOLUÇÃO PARA UM SUBCONJUNTO DE ITENS COM UMA CAPACIDADE PARCIAL. LOGO, O TEMPO E ESPAÇO SÃO PROPORCIONAIS A  $N * W$ .

- $W$  AFETA DIRETAMENTE O TAMANHO DA TABELA HORIZONTALMENTE:

- PARA CADA ITEM  $i$ , CALCULAMOS  $W+1$  SUBPROBLEMAS (CAPACIDADES DE 0 ATÉ  $W$ ). ISSO SIGNIFICA QUE A CADA AUMENTO DE 1 UNIDADE EM  $W$ , A QUANTIDADE DE CÁLCULOS EXTRAS É PROPORCIONAL A  $N$ . OU SEJA,  $W$  CRESCE HORIZONTALMENTE, MULTIPLICANDO O ESFORÇO FEITO POR CADA ITEM.

|           |   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|-----------|---|---|---|---|---|---|---|---|---|---|
| 0         | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 (1, 2)  | 0 |   |   |   |   |   |   |   |   |   |
| 2 (3, 4)  | 0 |   |   |   |   |   |   |   |   |   |
| 3 (5, 7)  | 0 |   |   |   |   |   |   |   |   |   |
| 4 (7, 10) | 0 |   |   |   |   |   |   |   |   |   |

$W$  é mais custoso que  $N$ !!



# SOLUÇÃO HEURÍSTICA - ALGORITMO GULOSO

- TÉCNICA:

- ESCOLHER SEMPRE O ITEM QUE OFERECE O MAIOR "BENEFÍCIO".

- COMPLEXIDADE:

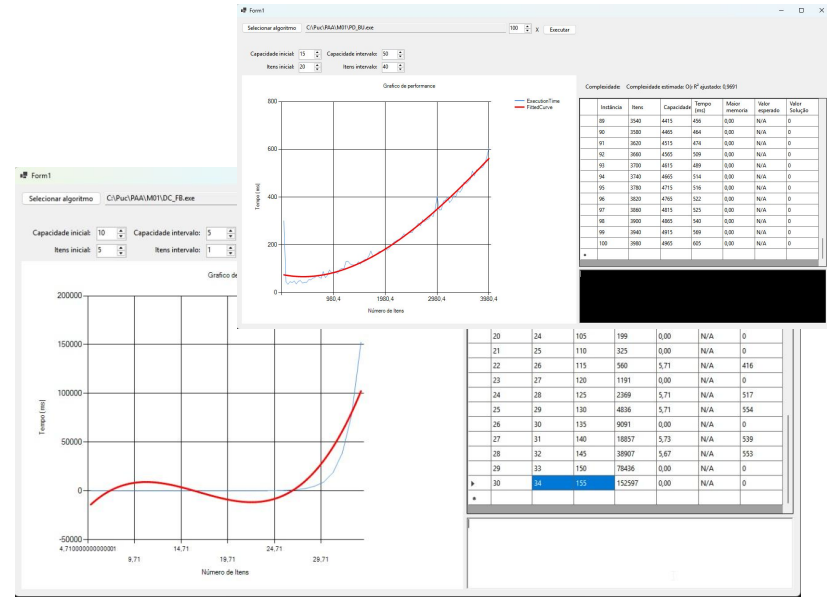
- $O(N \log N)$

```
ALGORITMO MochilaGulosa(capacidade, pesos, valores, n)
    1. Para cada item i, calcule o beneficio = valores[i] / pesos[i].
    2. Ordene os itens em ordem decrescente de beneficio.
    3. Inicialize a mochila como vazia e o valor_total = 0.
    4. Para cada item i na lista ordenada:
    5.     SE pesos[i] <= capacidade_restante ENTÃO
    6.         Adicione o item i à mochila.
    7.         valor_total += valores[i]
    8.         capacidade_restante -= pesos[i]
    9.     FIM SE
    10. RETORNE valor_total
FIM ALGORITMO
```

# FERRAMENTA DESENVOLVIDA (KPACK01\_ANALYZER)


## FUNCIONALIDADES:

- PERMITE EXECUTAR ALGORITMOS EXTERNOS (.EXE) DE FORMA PARAMETRIZADA.
- CRIA CENÁRIOS DE TESTE ALEATÓRIOS COM COMPLEXIDADE E TAMANHO CRESCENTES.
- MEDE AUTOMATICAMENTE O TEMPO DE EXECUÇÃO (MS) E O PICO DE USO DE MEMÓRIA (MB).
- PLOTA OS RESULTADOS EM TEMPO REAL

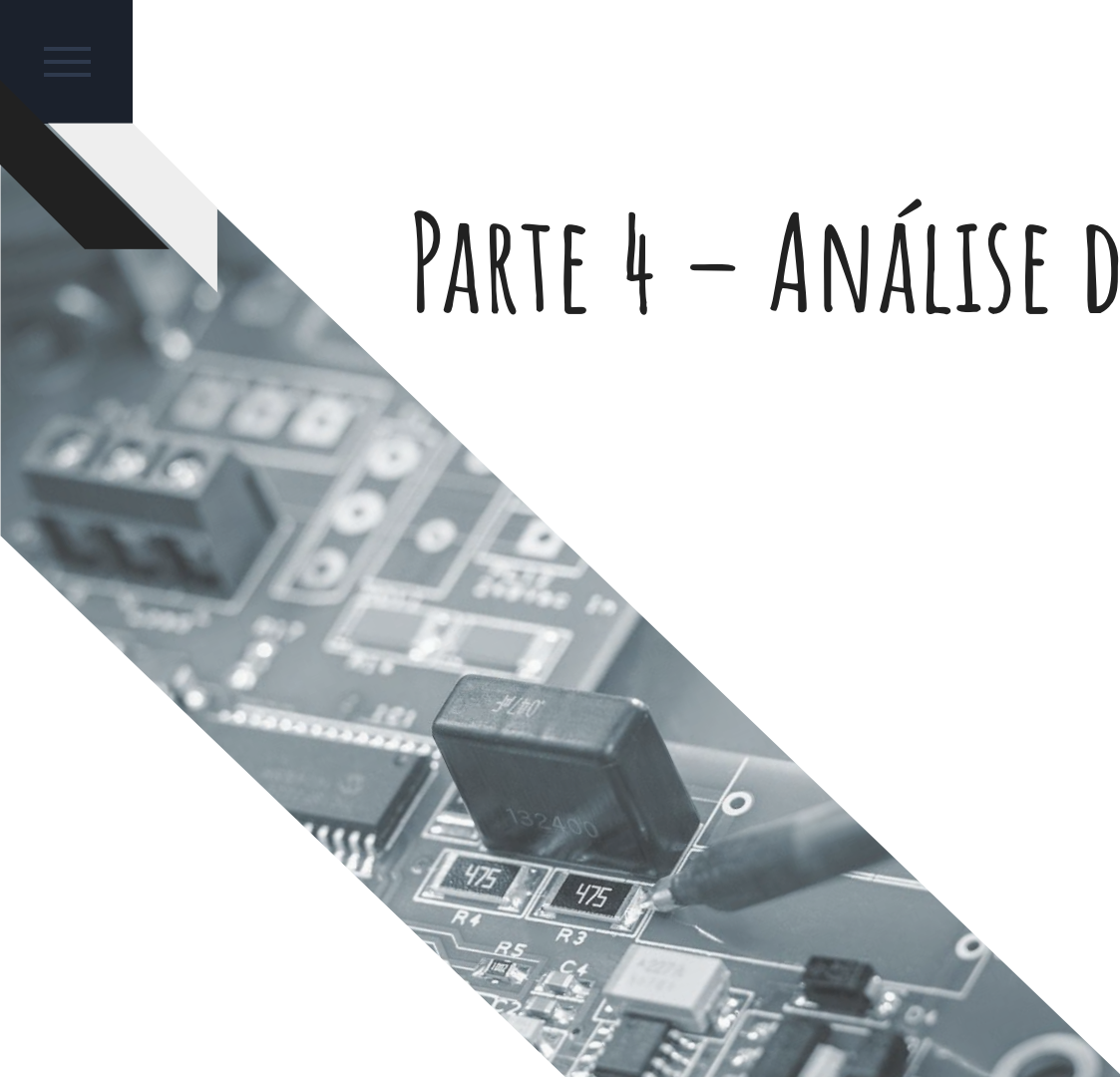


A ANÁLISE DE COMPLEXIDADE É EXPERIMENTAL, BASEADA EM DADOS DE PERFORMANCE, NÃO UMA PROVA FORMAL!

# COMPARAÇÃO ENTRE OS ALGORITMOS

|   | A   | B  | C  | D   |
|---|---|--|--|---|
|   | Comparação algoritmos  |  |  |   |
| 1 | Característica  | Força Bruta (Recursão)                     | Programação Dinâmica (PD)                      | Algoritmo Guloso (Heurística)   |
| 2 | Garante Solução Ótima?  | ✔ Sim                                      | ✔ Sim  | ✗ Não   |
| 3 | Complexidade de Tempo   | $O(2^n)$ (Exponencial)                     | $O(n \times W)$ (Pseudo-Polinomial)            | $O(n \log n)$ (Polinomial)  |
| 4 | Complexidade de Espaço  | $O(n)$ (Pilha de recursão)                 | $O(n \times W)$ ou $O(W)$                      | $O(1)$ ou $O(n)$  |
| 5 | Tipo de Abordagem   | Exato                                      | Exato  | Heurístico  |
| 6 | Cenário Ideal de Uso  | Instâncias muito pequenas; fins didáticos. | Padrão-ouro para a maioria dos casos práticos. | Instâncias muito grandes onde uma solução "boa e rápida" é aceitável. |
| 7 |   |  |  |   |

# PARTE 4 – ANÁLISE DOS RESULTADOS





# COMPORTAMENTO EXPERIMENTAL

- VALIDAÇÃO DA TEORIA:
  - OS RESULTADOS PRÁTICOS SE APROXIMARAM COM A COMPLEXIDADE TEÓRICA DE CADA ALGORITMO.
- FORÇA BRUTA - O "MURO EXPONENCIAL":
  - COMPLEXIDADE  $O(2^N)$  OBSERVADA NA PRÁTICA.
  - TORNOU-SE COMPUTACIONALMENTE INVIÁVEL PARA INSTÂNCIAS COM N MAIOR QUE 25-30.
- PROGRAMAÇÃO DINÂMICA - A EFICIÊNCIA PSEUDO-POLINOMIAL:
  - COMPLEXIDADE  $O(N \times W)$  VISÍVEL NOS TESTES.
  - APRESENTOU CRESCIMENTO LINEAR E CONTROLADO COM O AUMENTO DE N OU W.
- HEURÍSTICA GULOSA - PERFORMANCE POLINOMIAL:
  - COMPLEXIDADE  $O(N \log N)$  RESULTOU EM TEMPOS DE EXECUÇÃO QUASE INSTANTÂNEOS.
  - ORDENS DE MAGNITUDE MAIS RÁPIDO QUE AS SOLUÇÕES EXATAS.



# DISCUSSÃO

- FORÇA BRUTA: RELEVÂNCIA DIDÁTICA
  - EMBORA GARANTA A OTIMALIDADE, SUA INVIABILIDADE NA PRÁTICA DEMONSTRA POR QUE ALGORITMOS MAIS INTELIGENTES SÃO NECESSÁRIOS.
- PROGRAMAÇÃO DINÂMICA: A SOLUÇÃO PADRÃO-OURO
  - É A ABORDAGEM DE ESCOLHA PARA OBTER A SOLUÇÃO ÓTIMA EM TEMPO VIÁVEL, CONTANTO QUE A CAPACIDADE  $W$  NÃO SEJA EXCESSIVAMENTE GRANDE.
- HEURÍSTICA GULOSA: A VELOCIDADE TEM UM CUSTO
  - A RAPIDEZ EXTREMA DO ALGORITMO GULOSO VEM AO CUSTO DA INCERTEZA.

OBRIGADO!

**Autores:**

Alberto Magno Machado (632800)

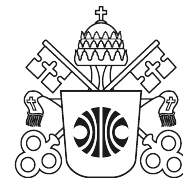
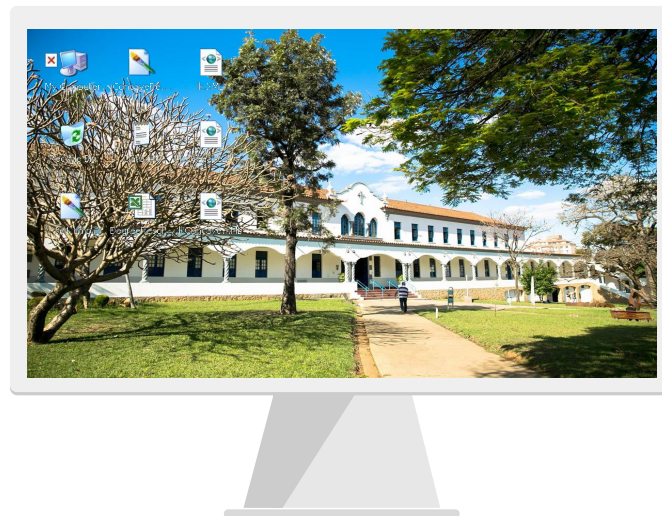
Leonardo Henrique Saraiva de Avelar ()

Josué Pereira Nogueira ()

**Professor:**

Prof. Walisson Ferreira de Carvalho

PUC Minas – Junho/2025



**PUC Minas**