


# Projeto e Análise de Algoritmos

## Aula 04 – Técnicas de Projeto de Algoritmos (Método Guloso)

Edirlei Soares de Lima  
<edirlei@iprj.uerj.br>

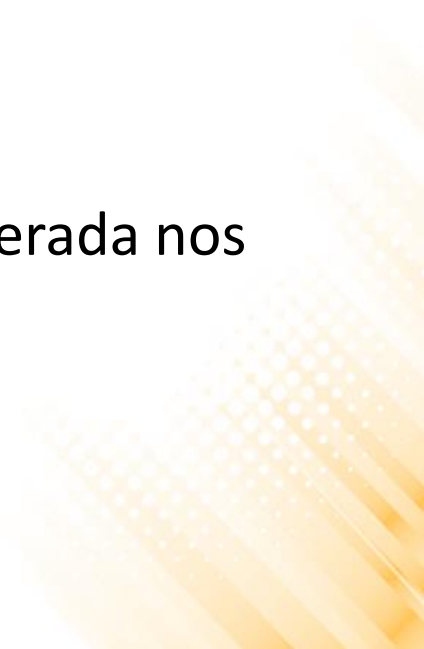
# Estratégias de Projeto de Algoritmos

- **Força Bruta (Brute Force)**
  - **Dividir e Conquistar (Divide and Conquer)**
  - Diminuir e Conquistar (Decrease and Conquer)
  - Transformar e Conquistar (Transform and Conquer)
  - Compromisso Tempo–Espaço (Space and Time Tradeoffs)
  - **Estratégia Gulosa (Greedy)**
  - **Programação Dinâmica (Dynamic Programming)**
  - Voltando Atrás (Backtracking)
  - Ramificar e Limitar (Branch and Bound)
  - Algoritmos Aproximados
- 

# Método Guloso

- **Ideia:** quando temos uma escolha a fazer, fazemos aquela que pareça ser a **melhor no momento**.
  - Ou seja, fazemos uma escolha **ótima local**, na esperança de obter uma solução **ótima global**.
- O método guloso sugere construir uma solução através de uma **sequência de passos**, cada um expandindo uma solução parcialmente construída até o momento, até uma solução completa para o problema ser obtida.
  - Métodos gulosos nem sempre garantem soluções ótimas, mas quando eles garantem, geralmente são as soluções mais simples e eficientes.

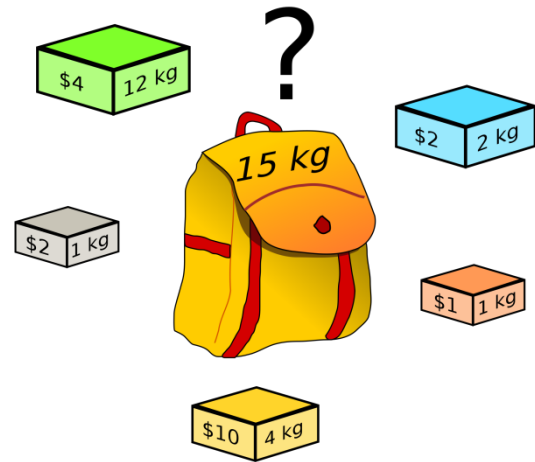
# Método Guloso

- Em cada passo de um algoritmo guloso, a escolha deve:
    - **Possível:** deve satisfazer as restrições do problema;
    - **Localmente ótima:** deve ser a melhor escolha local dentre todas as escolhas disponíveis naquele passo;
    - **Irreversível:** uma vez feita, ele não pode ser alterada nos passos subsequentes do algoritmo.
- 

# Exemplo 1: Problema da Mochila

- **Dados n itens:**

- Pesos:  $w_1, w_2, \dots, w_n$
- Valores:  $v_1, v_2, \dots, v_n$
- Uma mochila de capacidade  $W$



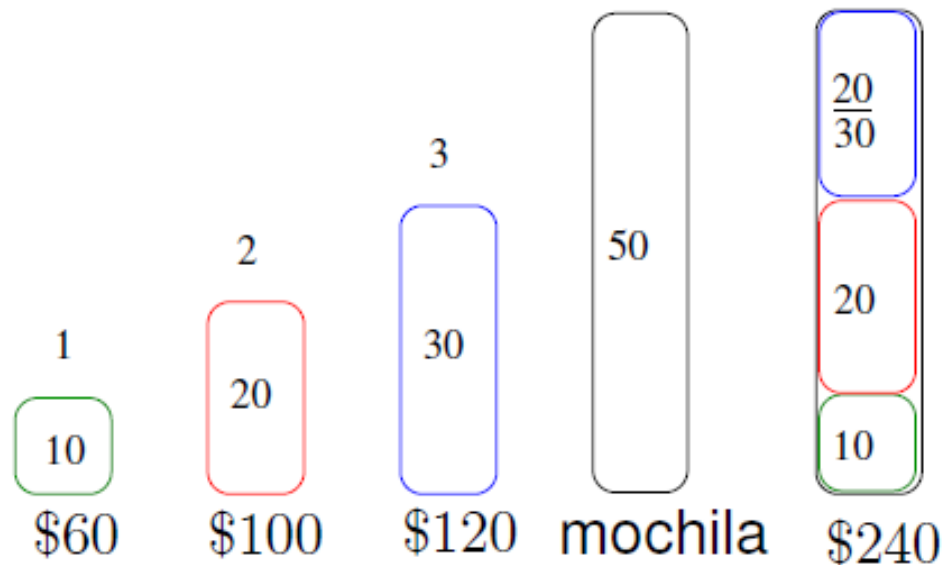
- **Problema:** encontrar o subconjunto mais valioso de itens que caibam dentro da mochila (Knapsack Problem).

# Exemplo 1: Problema da Mochila

- **Problema da Mochila 0-1:**
  - Um ladrão rouba uma loja que contém  $n$  itens: o item  $i$  tem peso  $w_i$  e vale  $v_i$ . Ele quer levar o maior valor possível em uma mochila de carga máxima  $W$ . Quais itens escolher?
- **Problema da Mochila Fracionada:**
  - Mesmo formulação anterior, mas agora ele pode carregar frações dos itens, ao invés da escolha binária (0-1).

# Exemplo 1: Problema da Mochila (Fracionada)

- **Algoritmo Guloso (ideia):**
  - Ordene os itens por valor/peso ( $v_i/w_i$ );
  - Começando em  $i = 1$ , coloque na mochila o máximo possível do item  $i$  que estiver disponível ;
  - Se puder levar mais, passe para o próximo item.



# Exemplo 1: Problema da Mochila (Fracionada)

- **Algoritmo Guloso:**

```
1. Mochila_Fracionada(item, valor, peso, n, totalmochila)
2.   MergeSort(item, valor, peso, n)
3.   carga ← 0
4.   i ← 1
5.   enquanto (carga < totalmochila) e (i <= n) faça
6.     se (peso[i] ≤ totalmochila - carga) então
7.       pegue todo o item[i]
8.       carga ← carga + peso[i]
9.     senão
10.      pegue (totalmochila - carga)/peso[i] do item[i]
11.      carga ← carga + (totalmochila - carga)/peso[i]
12.     i ← i + 1
13.  retorne itens pegados
```

**$O(n \log n)$**



# Exemplo 2: Seleção de Atividades

- **Seja:**
  - $S = \{a_1, \dots, a_n\}$ : conjunto de  $n$  atividades que podem ser executadas em um mesmo local. Exemplo: palestras em um auditório.
  - Para todo  $i = 1, \dots, n$ , a atividade  $a_i$  começa no instante  $s_i$  e termina no instante  $f_i$ , com  $0 < s_i < f_i < \infty$ .
  - As atividades  $a_i$  e  $a_j$  são ditas compatíveis se os intervalos  $[s_i, f_i]$  e  $[s_j, f_j]$  são disjuntos.
- **Problema:** Encontre em  $S$  um subconjunto de atividades mutuamente compatíveis que tenha tamanho máximo.

# Exemplo 2: Seleção de Atividades

- **Exemplo:**

$i$	1	2	3	4	5	6	7	8	9	10	11
$s_i$	1	3	0	5	3	5	6	8	8	2	12
$f_i$	4	5	6	7	8	9	10	11	12	13	14

- Pares de atividades incompatíveis:  $(a_1, a_2)$ ,  $(a_1, a_3)$
- Pares de atividades compatíveis:  $(a_1, a_4)$ ,  $(a_4, a_8)$
- Conjuntos máximos de atividades compatíveis:  
 $(a_1, a_4, a_8, a_{11})$  e  $(a_2, a_4, a_9, a_{11})$

# Exemplo 2: Seleção de Atividades

- **Algoritmo Guloso:**

```
1. Selecciona_Atividade(a, s, f, n)
2.   MergeSort(a, s, f, n) //ordenação crescente por  $f_i$ 
3.    $A \leftarrow \{a_1\}$ 
4.    $i = 1$ 
5.   Para  $m \leftarrow 2$  até  $n$  faça
6.     se  $s_m \geq f_i$  então
7.        $A \leftarrow A \cup \{a_m\}$ 
8.        $i \leftarrow m$ 
9.   retorne  $A$ 
```

**$O(n \log n)$**

# Método Guloso

- Outros algoritmos gulosos que veremos ao longo do curso:
  - Algoritmo de Prim (Árvores Geradoras Mínimas)
  - Algoritmo de Kruskal (Árvores Geradoras Mínimas)
  - Algoritmo de Dijkstra (Distâncias Mínimas)

# Comentários sobre o Método Guloso

- Vantagens:
  - Algoritmos simples;
  - Fácil implementação;
  - Quando garantem soluções ótimas, geralmente são os algoritmos mais simples e eficientes;
- Desvantagens:
  - Nem sempre garantem soluções ótimas locais;
  - Podem efetuar cálculos repetitivos;
  - Escolhe o caminho que, à primeira vista, é mais econômico;

# Exercícios

## **Lista de Exercícios 04 – Método Guloso**

<http://www.inf.puc-rio.br/~elima/paa/>



# Leitura Complementar

- Levitin. **Introduction to the Design and Analysis of Algorithms**, 3rd Edition, 2011.
- **Capítulo 9: Greedy Technique**

