

# CI1238 - Otimização

## Aula 16 - Algoritmos Gulosos

**Professor Murilo V. G. da Silva**

Departamento de Informática  
Universidade Federal do Paraná

2026 / Primeiro Semestre

## Algoritmos Gulosos

Árvore Geradora Mínima:

- ▶ Dado grafo ponderado conexo  $G$
- ▶ Encontrar subgrafo  $T$  de  $G$  tal que
  - ▶  $T$  é árvore (i.e., conexo e acíclico)
  - ▶  $V(T) = V(G)$
  - ▶ Soma dos pesos de  $E(T)$  é mínimo
- ▶ Esta árvore  $T$  é chamada de árvore geradora mínima (AGM)

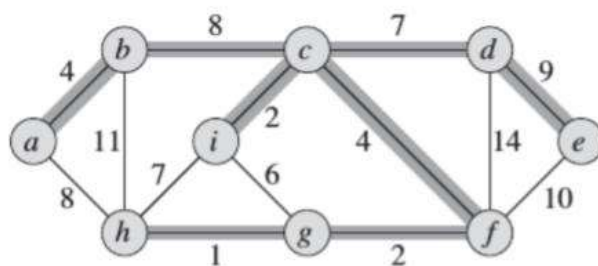


Figura: (Cormen, Leiserson, Rivest, and Stein, 2009, fig 23.1, p. 625)

# Algoritmos Gulosos

## Árvore Geradora Mínima: Algoritmo de Kruskal

### KRUSKAL ( $G$ )

- 1: Ordene as arestas de  $G$  e coloque em  $Q$
- 2: Inicialize  $T = (V, \emptyset)$
- 3: **while**  $|E(T)| < n - 1$  **do**
- 4:     Remova primeiro elemento de  $Q$  e coloque em  $e$
- 5:     **if**  $T + e$  é acíclico **then**
- 6:         Insira  $e$  em  $T$
- 7:     **else**
- 8:         Descarte  $e$
- 9:     **end if**
- 10: **end while**

# Algoritmos Gulosos

## Árvore Geradora Mínima: Algoritmo de Prim

### PRIM ( $G$ )

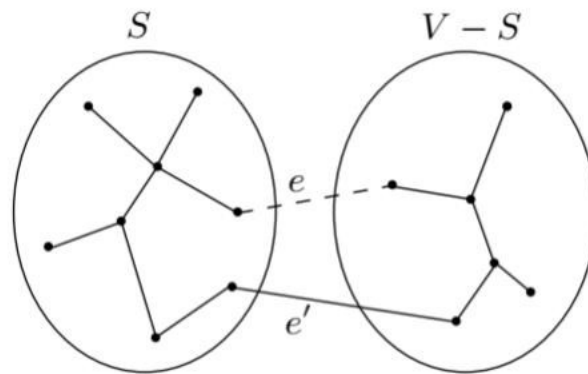
- 1: Fixe um vértice  $v$  (escolhido arbitrariamente)
- 2: Inicialize  $T = (\{v\}, \emptyset)$
- 3: **while**  $|E(T)| < n - 1$  **do**
- 4:     Escolha a aresta  $e = \{u, v\}$  de menor peso incidente a  $V(T)$
- 5:     **if**  $e$  tem as duas extremidades em  $V(T)$  **then**
- 6:         Descarte  $e$
- 7:     **else**
- 8:         Insira vértices  $u, v$  e aresta  $e$  em  $T$
- 9:     **end if**
- 10: **end while**

# Algoritmos Gulosos

Prova de ambos os algoritmos: [Propriedade do Corte](#)

**Teorema [DPV06]:** Seja  $G$  um grafo ponderado e seja  $X$  um conjunto de arestas tal que para alguma AGM  $T$  de  $G$ ,  $X \subseteq E(T)$ . Seja  $(S, V(G) \setminus S)$  uma partição de  $V(G)$  tal que toda aresta de  $X$  tenha suas duas pontas dentro de  $S$ . Seja  $e$  uma aresta de peso mínimo entre todas arestas com uma ponta em  $S$  e outra em  $V(G) \setminus S$ .

Então existe alguma AGM  $T^*$  de  $G$  tem  $X \cup \{e\} \subseteq E(T^*)$  (possivelmente  $T = T^*$ ).



Ideia da prova: A cada passo vamos construindo  $X$  tal que “ $X$  esteja no caminho” na construção de  $T$ . Se escolhermos  $e$  não faz parte de  $T$ , então existe  $e'$  que podemos trocar por  $e$  para obter  $T^*$ .

## Algoritmo guloso para Construção de Códigos de Huffman

- ▶ Códigos de Huffman: esquema usado em arquivos zip, gzip, png, pdf...

Considere uma sequência de 130 milhões de símbolos, alfabeto  $\Gamma = \{A, B, C, D\}$ .

Queremos codificar essa sequência em binário. Solução simples:

$A \mapsto 00$ ,  $B \mapsto 01$ ,  $C \mapsto 10$ ,  $D \mapsto 11$ .

**Terminologia:** “00 é o código para o símbolo A”

Como cada símbolo utiliza códigos de 2 bits, precisamos de 260 milhões de bits.

**É possível fazer melhor?**

# Algoritmo guloso para Construção de Códigos de Huffman

**Caso comum:** Os símbolos não aparecem com a mesma frequência.

Símbolo	Frequência
<i>A</i>	70 milhões
<i>B</i>	3 milhões
<i>C</i>	20 milhões
<i>D</i>	37 milhões

**Observação:**

- ▶ *A* corresponde a mais da metade das ocorrências.
- ▶ *B* é extremamente raro.

Uma codificação mais interessante poderia ser:

$A \mapsto 0$   
 $B \mapsto 100$   
 $C \mapsto 101$   
 $D \mapsto 11$

# Algoritmo guloso para Construção de Códigos de Huffman

Nem todo conjunto de códigos de comprimentos diferentes funciona. Exemplo:

$A \mapsto 0$ ,     $B \mapsto 01$ ,     $C \mapsto 11$ ,     $D \mapsto 001$ .

A string binária 001 corresponde a quais símbolos? A string *AB* ou a string *D*?

**Código Livre de Prefixos:** Nenhum código é prefixo de outro. → sempre funciona!

- ▶ A codificação do slide anterior é um exemplo
- ▶ O **Algoritmo de Huffman** resolve o problema de encontrar uma codificação ótima

Problema que o algoritmo resolve:

**Entrada:** String de certo alfabeto

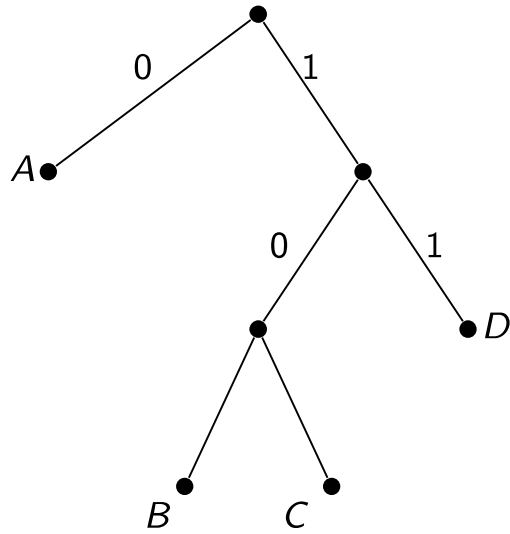
**Saída:** Codificação binária ótima

Variação do problema: Entrada é lista de símbolos com respectivas frequências médias

# Algoritmo guloso para Construção de Códigos de Huffman

Usando uma árvore binária para representar a codificação:

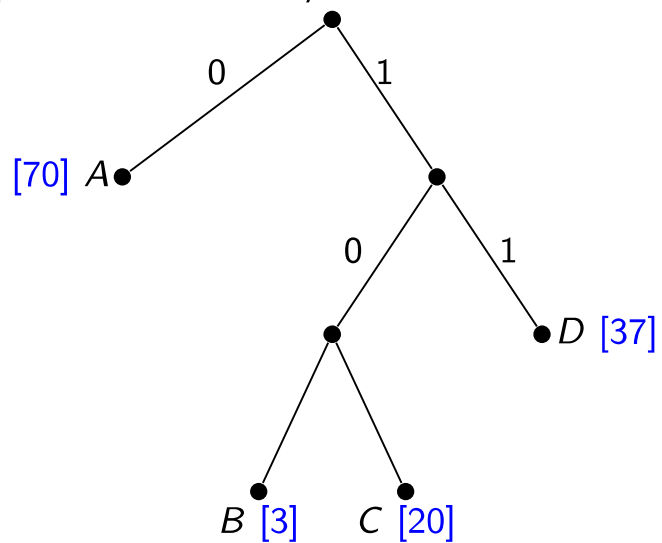
Símbolo	String Binária
A	0
B	100
C	101
D	11



# Algoritmo guloso para Construção de Códigos de Huffman

Usando uma árvore binária para representar a codificação:

Símbolo	String Binária
A	0
B	100
C	101
D	11

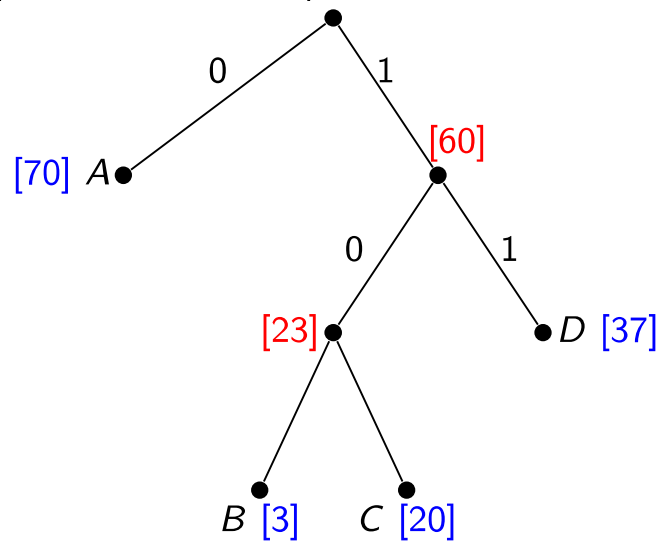


Incluindo as frequências dos símbolos na árvore

# Algoritmo guloso para Construção de Códigos de Huffman

Usando uma árvore binária para representar a codificação:

Símbolo	String Binária
A	0
B	100
C	101
D	11



Incluindo as frequências dos símbolos na árvore e frequências nos nós internos.

# Algoritmo guloso para Construção de Códigos de Huffman

Formalizando um pouco:

Suponha que  $\Gamma = \{1, 2, \dots, n\}$  e os símbolos  $1, 2, \dots, n$  possuem frequências

$$f_1, f_2, \dots, f_n.$$

Queremos: Árvore de codificação  $T$  com símbolos nas folhas tal que o tamanho da string codificada é mínimo.

Se o símbolo  $i$  aparece em uma folha de profundidade  $d_i$ , então ele contribui com  $f_i \cdot d_i$  bits no tamanho da string codificada em binário

Custo da árvore:

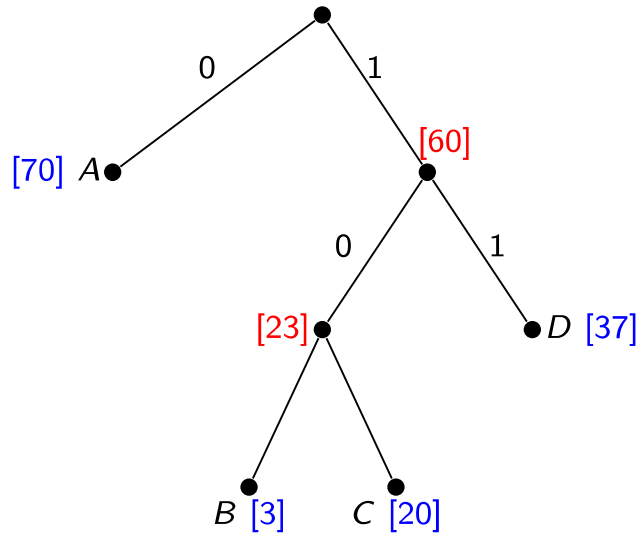
$$\text{custo}(T) = \sum_{i=1}^n f_i d_i.$$

- ▶ Importante: vamos incluir em  $T$  as frequências (incluindo nós internos)
- ▶ Decodificando uma string binária: O número de vezes que visitamos cada nó é exatamente a frequência associada ao nó

# Algoritmo guloso para Construção de Códigos de Huffman

- ▶ Decodificando uma string binária: O número de vezes que visitamos cada nó é exatamente a frequência associada ao nó

Símbolo	String Binária
A	0
B	100
C	101
D	11



- ▶ Observação: O custo de  $T$  também é igual à soma das frequências  $f_v$  de todos os nós  $v$  (internos e folhas) de  $T$ , exceto a raiz.

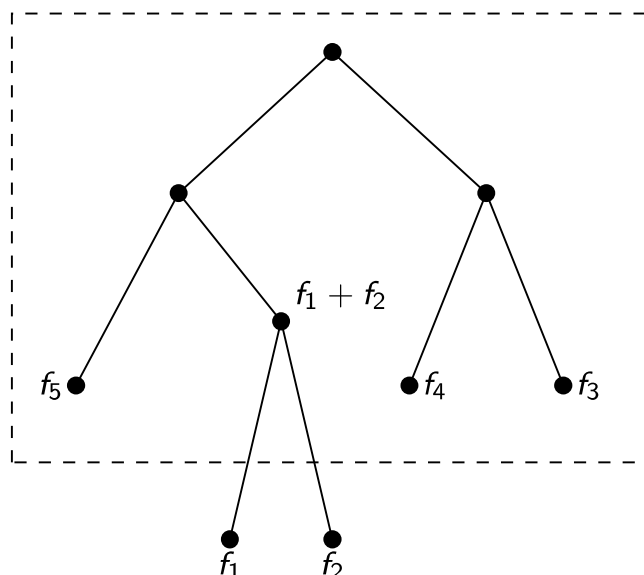
$$\text{i.e., } \text{custo}(T) = \sum_{i=1}^n f_i d_i = \sum_{v \in T} f_v$$

- ▶ **Fato crucial:** Os dois símbolos com menores frequências devem aparecer nas folhas mais profundas da árvore ótima.  
**Prova:** Caso contrário poderíamos trocar um desses símbolos por outro de frequência maior mais abaixo em  $T$  e obter  $T'$  tal que  $\text{custo}(T') < \text{custo}(T)$ .

# Algoritmo guloso para Construção de Códigos de Huffman

## Algoritmo Guloso:

1. Encontre as duas menores frequências  $f_1$  e  $f_2$ .
2. Criar um novo nó pai para elas.
3. Atribuir ao novo nó a frequência  $f_1 + f_2$ .
4. Resolver recursivamente o problema para a instância  $(f_1 + f_2), f_3, f_4, \dots, f_n$



Complexidade:  $O(n \log n)$  (usando um *min heap*)

# Algoritmo Guloso para Cobertura por Conjuntos

Problema típico: Cidades que devem ser atendidas por algum serviço:

→ Não há necessidade de instalar serviço em cada uma das cidades, pois algumas cidades são próximas uma das outras. Qual a menor quantidade de cidades?



$$B = \{a, b, c, d, e, f, g, h, i, j, k\}$$

$$S_a = \{a, b, d, e, h, i, k\}, S_b = \{a, b, c, d\}, S_c = \{b, c, d\}, \dots, S_k = \{a, h, i, j, k\}$$

**Cobertura por Conjuntos (CC):**

- ▶ **Entrada:** Um conjunto  $B$  de tamanho  $n$  e  $m$  subconjuntos  $S_1, \dots, S_m \subseteq B$ .
- ▶ **Saída:** Uma lista de  $k \leq m$  subconjuntos  $S_i$  de forma que a união dos  $k$  conjuntos inclua todos  $n$  elementos de  $B$  e  $k$  seja mínimo.

Algoritmo Guloso: A cada passo escolha o conjunto  $S_i$  que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos.

# Algoritmo Guloso para Cobertura por Conjuntos

Algoritmo Guloso: A cada passo escolha o conjunto  $S_i$  que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos.

- ▶ Seja  $n$  o tamanho de  $B$  e  $k$  o tamanho da solução ótima
- ▶ Estratégia gulosa não funciona (vimos isto no exemplo do slide anterior)
- ▶ Mas a estratégia garante uma solução de tamanho no máximo  $k \cdot \ln n$

Veremos isso agora.

# Algoritmo Guloso para Cobertura por Conjuntos

## Cobertura por Conjuntos (CC):

- ▶ **Entrada:** Um conjunto  $B$  de tamanho  $n$  e  $m$  subconjuntos  $S_1, \dots, S_m \subseteq B$ .
- ▶ **Saída:** Uma lista de  $k$  subconjuntos  $S_i$  de forma que a união dos  $k$  conjuntos inclua todos  $n$  elementos de  $B$  e  $k$  seja mínimo.

**Algoritmo Guloso:** A cada passo escolha o conjunto  $S_i$  que contenha mais elementos ainda não inclusos nos conjuntos já escolhidos.

**Teorema:** Se a solução ótima tem tamanho  $k$ , então o algoritmo guloso devolve uma solução com  $k \cdot \ln n$  elementos, onde  $|B| = n$ .

- ▶ **Fato 1:** Seja  $n_i$  o número de elementos descobertos depois de  $i$  passos ( $n_0 = n$ ). A cada passo uma fração de pelo menos  $1/k$  dos elementos restantes é coberto

**Passo 1:** Sobram  $n_1 \leq (1 - 1/k)n_0$  elementos;

**Passo 2:** Sobram  $n_2 \leq (1 - 1/k)n_1 \leq (1 - 1/k)^2 n_0$  elementos;

**Passo 3:** Sobram  $n_3 \leq (1 - 1/k)n_2 \leq (1 - 1/k)^3 n_0$  elementos;

⋮

**Passo  $t$ :** Sobram  $n_t \leq (1 - 1/k)n_{t-1} = (1 - 1/k)^t n_0 = (1 - 1/k)^t n$  elementos;

Vamos provar que no passo  $t = \ln n \cdot k$  todos elementos foram escolhidos

**Fato 2:**  $\forall x \neq 0, 1 - x < e^{-x}$ . Consequência: Como  $\frac{1}{k} \neq 0$ , então  $1 - \frac{1}{k} < e^{-\frac{1}{k}}$

$$n_t \leq (1 - 1/k)^t \cdot n < (e^{-1/k})^t \cdot n = (e^{-1/k})^{\ln n \cdot k} \cdot n = e^{\frac{-\ln n \cdot k}{k}} \cdot n = \frac{1}{e^{\ln n}} \cdot n = \frac{1}{n} \cdot n = 1$$

# Algoritmo Guloso para Cobertura por Conjuntos

## Cobertura por Conjuntos (CC):

- ▶ **Entrada:** Um conjunto  $B$  de tamanho  $n$  e  $m$  subconjuntos  $S_1, \dots, S_m \subseteq B$ .
- ▶ **Saída:** Uma lista de  $k$  subconjuntos  $S_i$  de forma que a união dos  $k$  conjuntos inclua todos  $n$  elementos de  $B$  e  $k$  seja mínimo.

**Caso particular do problema:** Antes de vermos um algoritmo guloso para CC, vamos ver a relação deste problema com o problema da cobertura por vértices

## Cobertura por Vértices (CV)

- ▶ **Entrada:** Um grafo  $G = (V, E)$ .
- ▶ **Saída:** Conjunto  $S$  de vértices de tamanho mínimo tal que  $\forall e \in E$ , a aresta  $e$  tem pelo menos uma ponta em  $S$

Note que CV é um caso particular de CC, pois dado  $G = (V, E)$  com vértices  $v_1, \dots, v_n$

- ▶ Faça  $B = E(G)$  e  $S_i = \partial(v_i), v_i \in V$   
(ou seja,  $S_i$  é o conjunto de arestas incidentes a  $v_i$ )
- ▶ Portanto algoritmo guloso (escolher vértice de maior grau) encontra cobertura de tamanho  $k \cdot \ln |V|$  (sendo  $k$  o tamanho da cobertura ótima)
- ▶ Obs: No caso particular de existem algoritmos com aproximações melhores (e.g., usando *PL relaxado* a garantia de no máximo  $2k$  vértices), mas para o caso geral do CC uma  $\ln n$ -aproximação é a melhor possível (no cenário  $\neq NP$ ).

# Algoritmos Gulosos: Considerações finais

Em geral, o seguinte pode ser dito sobre algoritmos gulosos:

- ▶ Quando funcionam, geralmente os algoritmos são simples e eficientes.
- ▶ A prova de corretude pode ser difícil.
  
- ▶ Para muitos problemas não funcionam e não têm garantias de aproximação.

Considere o problema da clique máxima.

O que seria um algoritmo guloso para este problema?

- ▶ Escolher vértices de maior grau possível?
- ▶ Excluir vértices de menor grau possível?

**Exercício:** Mostre que ambas estratégias não funcionam.

Considere o problema da coloração de grafos (cores 1, 2, 3, ...).

O que seria um algoritmo guloso para este problema?

- ▶ Para cada vértice, escolher a “menor cor” disponível.

**Exercício:** Mostre que este algoritmo guloso não funciona.