Aula 6

- Algoritmos Gananciosos
  - Problema da Mochila Fracionária
  - Problema da Seleção de Atividades
  - Códigos de Huffman

# Algoritmos Greedy - Problema da Mochila Fraccionária

Input:

- Mochila com capacidade $W$
- $n$ items com pesos $w_1, \dots, w_n$ e valores $v_1, \dots, v_n$

Objectivo:

- Que items devemos colocar na mochila de modo a transportarmos o maior valor possível, sabendo que podemos transportar uma quantidade fraccionária de cada item?

$$\max \sum_{i=1}^{n} v_i \cdot x_i$$

$$\forall\, 1 \le i \le n.\ 0 \le x_i \le 1$$

$$\sum_{i=1}^{n} x_i \cdot w_i \le W$$

- $x_i \Rightarrow$ fracção do produto $i$ transportada

# Algoritmos Greedy - Problema da Mochila Fraccionária

Input:

- Mochila com capacidade $V$
- $n$ items com pesos $w_1, \ldots, w_n$ e valores $v_1, \ldots, v_n$

Escolha Greedy: Colocar na mochila a maior quantidade possível do item com maior valor por unidade de peso.

- Calcular para cada item $i$: $v_i/w_i$
- Determinar o item $k$ tal que:

$$v_k/w_k = \max \{ v_i/w_i \mid 1 \leq i \leq n \}$$

- Subproblema:

$$(x_k, V') = \begin{cases} (1, V - w_k) & \text{se } V > w_k \\ (w_k/V, 0) & \text{c.c.} \end{cases}$$

# Algoritmos Greedy - Problema da Mochila Fracionária


$60/10 = 6$ $\quad \frac{100}{20} = 5$ $\quad \frac{120}{30} = 4$

① $x_1 = 1$, $W' = 50 - 10 = 40$

② $x_2 = 1$, $W' = 40 - 20 = 20$

③ $x_3 = \frac{20}{30} = \frac{2}{3}$, $W' = 0$

$C = 60 + 100 + \frac{2}{3} \cdot 120$

$= 60 + 100 + 80$

$= 240$

# Algoritmos Greedy - Problema da Mochila Fraccionária

Fractional Knapsack ($\vec{w}$, $\vec{v}$, $W$)

let $\vec{x}[1..n]$ be a new array initialized to 0
let $W' = W$

for $k = 1$ to $n$
  if $\vec{w}[k] < W'$
    $\vec{x}[k] := 1$
    $W' := W' - \vec{w}[k]$
  else
    $\vec{x}[k] := W' / \vec{w}[k]$
    return $\vec{x}$
return $\vec{x}$

- Admitimos que os arrays $\vec{w}$ e $\vec{v}$ se encontram ordenados por ordem decrescente de valor por unidade de peso: $v_k / w_k$

# Algoritmos Greedy - Problema da Mochila Fraccionária

FractionalKnapsack ($\vec{w}, \vec{v}, W$)

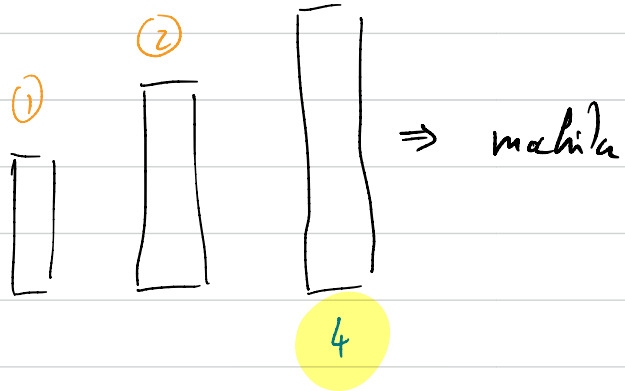
```
let x[1..n] be a new array initialized to 0
let W' = W

for k = 1 to n
    if w[k] < W'
        x[k] := 1
        W' := W' - w[k]
    else
        x[k] := W' / w[k]
        return x
return x
```

- Análise de Complexidade

$O(n)$

# Problema da Mochila NÃO Fraccionária

- Não é permitido transportar quantidades fraccionárias




- Exemplo: Escolha greedy não conduz à solução óptima

① $w_1 = 1$ $v_1 = 3$ $3/1 = 3 > 4/4 = 1$

② $w_2 = 4$ $v_2 = 4$

Escolha greedy ⇒ ① ✗

# Problema de Selecção de Actividades

## Definição [Problema de Selecção de Actividades]

**Input:** $S = \{a_1, \ldots, a_n\}$
$a_i$ executa no intervalo $[s_i, f_i)$

**Output:** $S^* \subseteq S$ de tamanho máximo tal que todas as actividades em $S^*$ são mutuamente compatíveis.

- Duas actividades $a_i$ e $a_j$ são mutuamente compatíveis se $[t_i, f_i) \cap [t_j, f_j) = \emptyset$

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 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

- Actividades Mutuamente Compatíveis:

- Actividades Mutuamente Incompatíveis:

# Problema de Selecção de Actividades

## Definição [Problema de Selecção de Actividades]

**Input:** $S = \{a_1, \ldots, a_n\}$
$a_i$ executa no intervalo $[s_i, f_i)$

**Output:** $S^* \subseteq S$ de tamanho máximo tal que todas as actividades em $S^*$ são mutuamente compatíveis.

- Duas actividades $a_i$ e $a_j$ são mutuamente compatíveis se $[t_i, f_i) \cap [t_j, f_j) = \emptyset$

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 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

- Actividades Mutuamente Compatíveis:
  $(1, 4), \ldots$

- Actividades Mutuamente Incompatíveis:
  $(10, 11), (1, 2), \ldots$

# Selecção de Actividades

**Objectivo:** Determinar o maior conjunto de actividades mutuamente compatíveis.

**Escolha Greedy:** Dado um conjunto de actividades $S$, determinar uma actividade a que pertence a um conjunto máximo de actividades mutuamente compatíveis.

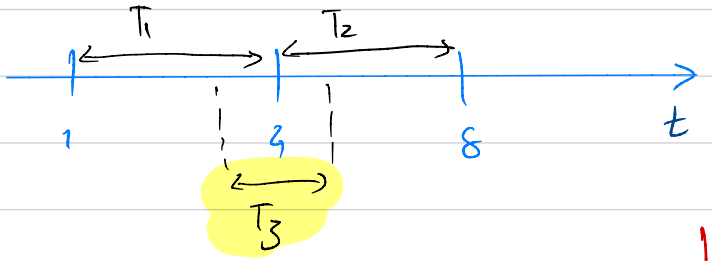
**Estratégia 1:** Seleccionar a actividade de menor duração.

# Selecção de Actividades

**Objectivo:** Determinar o maior conjunto de actividades mutuamente compatíveis.

**Escolha Greedy:** Dado um conjunto de actividades $S$, determinar uma actividade a que pertence a um conjunto máximo de actividades mutuamente compatíveis.

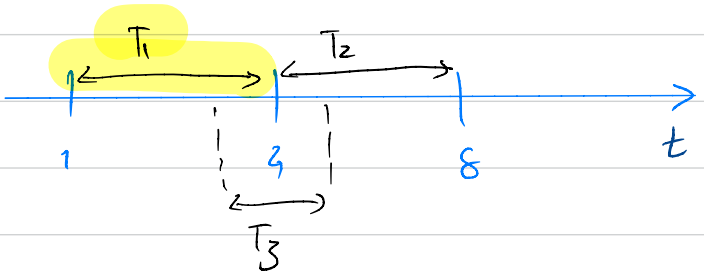
**Estratégia 1:** Seleccionar a actividade de menor duração.




Wrong!

# Selecção de Actividades

Estratégia 2: Seleccionar a actividade com menor tempo de fim.



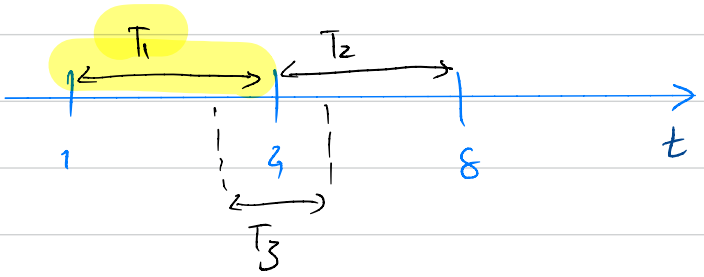

**Lema [Selecção de Actividades - Escolha Greedy]**

Seja $S$ um conjunto de actividades e seja $a_0$ a actividade com menor tempo de fim em $S$; então $a_0$ pertence a um conjunto máximo de actividades mutuamente compatíveis $S^*$.

Prova:

# Selecção de Actividades

Estratégia 2: Seleccionar a actividade com menor tempo de fim.




**Lema [Selecção de Actividades - Escolha Greedy]**

Seja $S$ um conjunto de actividades e seja $a_0$ a actividade com menor tempo de fim em $S$; então $a_0$ pertence a conjunto máximo de actividades mutuamente compatíveis $S^*$.

**Prova:**

- Seja $S^*$ um conjunto máximo de actividades mutuamente compatíveis.
- Se $a_0 \in S^*$, não há mais a provar.
- Suponhamos que $a_0 \notin S^*$; seja $a_0'$ a actividade com menor tempo de fim em $S^*$. Definimos o seguinte conjunto:

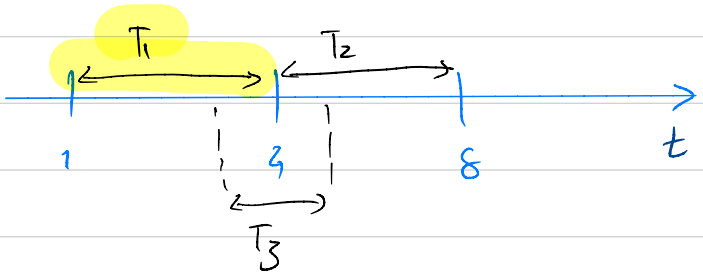
  $\hat{S}^* = (S^* \setminus \{a_0'\}) \cup \{a_0\}$

  - Por definição, temos que $|\hat{S}^*| = |S^*|$

    Resta provar todas as actividades em $\hat{S}^*$ são mutuamente compatíveis $\Leftrightarrow$ $a_0$ é compatível com todas as actividades em $S^* \setminus \{a_0'\}$.

# Selecção de Actividades

Estratégia 2: Seleccionar a actividade com menor tempo de fim.




**Lema [Selecção de Actividades - Escolha Greedy]**

Seja $S$ um conjunto de actividades e seja $a_0$ a actividade com menor tempo de fim em $S$; então $a_0$ pertence a conjunto máximo de actividades mutuamente compatíveis $S^*$.

Prova:

Tb provar:

$a_0$ é compatível com todas as actividades em $S^* \setminus \{a_0'\}$.

* Tome-se $a_k \in S^* \setminus \{a_0'\}$.

$\Rightarrow s_k \geq f_0'$

$\Rightarrow f_0' \geq f_0$

$\Rightarrow s_k \geq f_0 \Rightarrow [s_0, f_0) \cap [s_k, f_k) = \emptyset$

∎

# Selecção de Actividades - Algoritmo Greedy

```
Activity Selector (s, f)
    n = s.length
    A := {a_1}
    k := 1
    for i=2 to n
        if s[i] > f[k]
            A := A ∪ {a_k}
            k := i
    return A
```

// As actividades encontram-se inicialmente ordenadas por tempo de fim

Análise de Complexidade

- $O(n)$
  - ↳ Porque admitimos que as actividades se encontram desde o início convenientemente ordenadas.

# Selecção de Actividades - Algoritmo Greedy

```
Activity Selector (s, f)
    n = s.length
    A := {a_1}
    k := 1
    for i=2 to n
        if s[i] > f[k]
            A := A ∪ {a_k}
            k := i
    return A
```

// As actividades encontram-se inicialmente ordenadas por tempo de fim

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 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

# Selecção de Actividades - Algoritmo Greedy

```
Activity Selector (s, f)
    n = s.length
    A := {a_1}
    k := 1
    for i=2 to n
        if s[i] > f[k]
            A := A ∪ {a_k}
            k := i
    return A
```

// As actividades encontram-se inicialmente ordenadas por tempo de fim

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 | 9 | 9 | 10 | 11 | 12 | 14 | 16 |

$A = \{a_1, a_4, a_8, a_{11}\}$

# Códigos de Huffman

**Definição [Código Binário]**

Dado um alfabeto $C$, um código binário sobre $C$ é uma função $BC : C \to \{0,1\}^*$, que mapeia os elementos de $C$ em sequências de 0's e 1's.

## Tipos de Código

- Códigos de Comprimento Fixo (Fixed length Code)

  Todos os elementos de $C$ são mapeados em sequências de 0's e 1's do mesmo tamanho.

  Fixed-length Code
  a - 00
  b - 01
  c - 10

- Códigos de Comprimento Variável (Variable Length Code)

  Códigos que associam sequências de 0's e 1's de comprimento diferente a diferentes elementos de $C$

  Variable-Length Code
  a - 0
  b - 10
  c - 11

Questão: Quando é que faz sentido usar códigos de comprimento variável?

# Códigos de Huffman

- Qual o problema com o seguinte código?

$C = \{a, b, c, d\}$

a - 0
b - 01
c - 10
d - 1

# Códigos de Huffman

- Qual o problema com o seguinte código?

$C = \{a, b, c, d\}$

a - 0
b - 01
c - 10
d - 1

- Como descodificar 0010?
  - a a d a
  - a b a
  - a a c
  - ...

**Definição [Códigos Livres de Prefixo (Prefix-free Codes)]**
Um código binário Code sobre um alfabeto $\mathcal{C}$ diz-se livre de prefixo sse para quaisquer elementos $c_1, c_2 \in \mathcal{C}$ nem $Code(c_1)$ é prefixo de $Code(c_2)$ nem vice-versa.

Código 1:
a - 0
b - 10
c - 11

Código 2:
a - 0
b - 01
c - 10
d - 1

# Códigos de Huffman

- Qual o problema com o seguinte código?

$C = \{a, b, c, d\}$

a - 0
b - 01
c - 10
d - 1

- Como descodificar 0010?
  - a a d a
  - a b a
  - a a c
  - ...

**Definição [Códigos Livres de Prefixo (Prefix-free Codes)]**
Um código binário Code sobre um alfabeto $\mathcal{C}$ diz-se livre de prefixo sse para quaisquer elementos $c_1, c_2 \in \mathcal{C}$ nem $Code(c_1)$ é prefixo de $Code(c_2)$ nem vice-versa.

Código 1:
a - 0
b - 10
c - 11
livre de prefixo

Código 2:
a - 0
b - 01
c - 10
d - 1
Não livre de prefixo

# Códigos de Prefixo Óptimos

- Cada símbolo do alfabeto de input C é associado a uma frequência f.

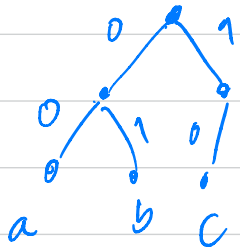
- Goal: Construir um código que minimize o nº médio de bits necessários para representar cada símbolo

# Códigos Binários ⟷ Árvores Binárias

- Fixed-length code
  - a - 00
  - b - 01
  - c - 10
- Variable-length code (Prefix-free)
  - a - 0
  - b - 10
  - c - 11
- Variable-length code (Non-Prefix-free)
  - a - 1
  - b - 10
  - c - 11

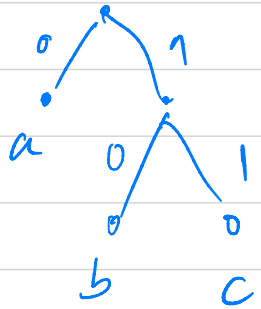
# Códigos Binários ⟷ Árvores Binárias

- Fixed-length code
  - a - 00
  - b - 01
  - c - 10



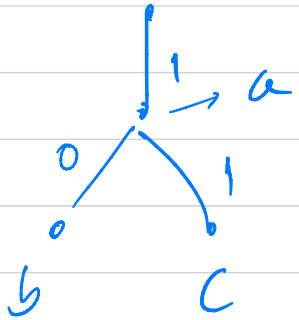

Facto 1: Um código binário é livre de prefixo sse na sua representação em árvore binária os únicos nós anotados com elementos de $C$ são as folhas.

- Variable-Length Code (Prefix-free)
  - a - 0
  - b - 10
  - c - 11




Facto 2: Tamanho do código necessário para codificar o elemento $c \in C$ é dado pela profundidade da folha associada a $c$.

- Variable-Length Code (Non-Prefix-free)
  - a - 1
  - b - 10
  - c - 11



# Códigos de Prefixo Óptimos

- Cada símbolo do alfabeto de input $C$ é associado a uma frequência $f$.
- Objectivo (versão 1): Construir um código que minimize o nº médio de bits necessários para representar cada símbolo

**Definição [Custo de Código Binário]**

Seja $T$ uma árvore binária q representa um código binário sobre um alfabeto $C$ com frequências $f$; o custo de $T$ é definido como se segue:

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c)$$

Facto: A profundidade de $c$ na árvore $T$ corresponde ao tamanho do código que $T$ associa a $c$.

- Objectivo (versão 2): Dado um alfabeto $C$ e uma função de frequências $f$, calcular a árvore $T$ q representa um código binário de $C$ com menor valor de $B(T)$.

# Códigos de Prefixo Ótimos

Definição [Custo de Código Binário]

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c)$$

| C | f | FLC | VLC |
|---|---|---|---|
| A | 60% | 00 | 0 |
| B | 25% | 01 | 10 |
| C | 10% | 10 | 110 |
| D | 5% | 11 | 111 |

* Comprimento Médio por Caracter

- FLC:

- VLC:

# Códigos de Prefixo Óptimos

Definição [Custo de Código Binário]

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c)$$

| C | f | FLC | VLC |
|---|---|---|---|
| A | 60% | 00 | 0 |
| B | 25% | 01 | 10 |
| C | 10% | 10 | 110 |
| D | 5% | 11 | 111 |

* Comprimento Médio por Caracter

- FLC: 2

- VLC: $1 \times 0.6 + 2 \times 0.25 + 3 \times 0.1 + 3 \times 0.05$

$= 1.55$

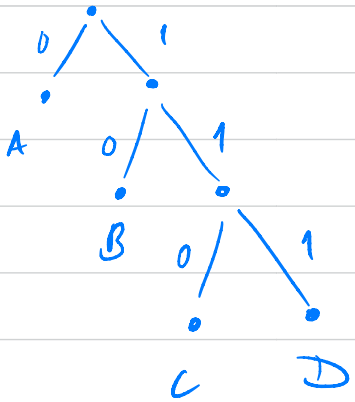
# Códigos de Prefixo – Descodificação

Exemplo:

A - 0
B - 10
C - 110
D - 111

- Queremos descodificar: 0110 111

① Árvore Binária q representa o código




② Descodificar: seguir os arcos da árvore a partir da raiz quando chegamos a uma folha voltamos à raiz.

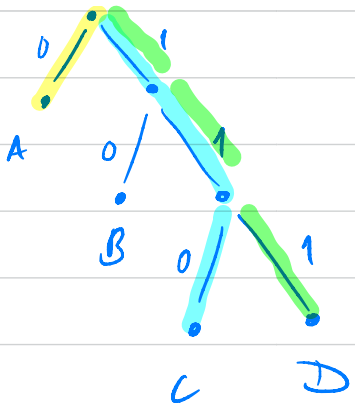
# Códigos de Prefixo – Descodificação

Exemplo:

A - 0
B - 10
C - 110
D - 111

- Queremos descodificar: 0110111

① Árvore Binária q representa o código




② Descodificar: seguir os arcos da árvore a partir da raiz quando chegamos a uma folha voltamos à raiz.

A C D

# Códigos de Prefixo - Sumário

- Códigos binários podem ser representados por árvores binárias
- Códigos livres de prefixo correspondem a árvores binárias cujas folhas estão anotadas com os elementos do alfabeto $C$.
- $f(c)$ - frequência do elemento $c$ ($c \in C$)
- $d_T(c)$ - profundidade de $c$ na árvore $T$
  $\Rightarrow$ comprimento do código q $T$ associa a $c$
- Custo do código representado por $T$:

$$B(T) = \sum_{c \in C} f(c) \cdot d_T(c)$$

# Algoritmo de Huffman

```
Huffman(C)
    n := |C|
    let Q be a min-priority queue with content C
    for i = 1 to n-1
     z := New Node()
      x := ExtractMin(Q)
      y := ExtractMin(Q)
      z.left := x;
      z.right := y;
      z.freq := x.freq + y.freq;
      InsertKey(Q, z)
    return ExtractMin(Q)
```

# Algoritmo de Huffman

```
Huffman(C)
    n := |C|
    let Q be a min-priority queue with content C
    for i = 1 to n-1
      z := New Node()
       x := ExtractMin(Q)
       y := ExtractMin(Q)
       z.left := x;
       z.right := y;
       z.freq := x.freq + y.freq;
       Insert Key(Q, z)
    return ExtractMin(Q)
```

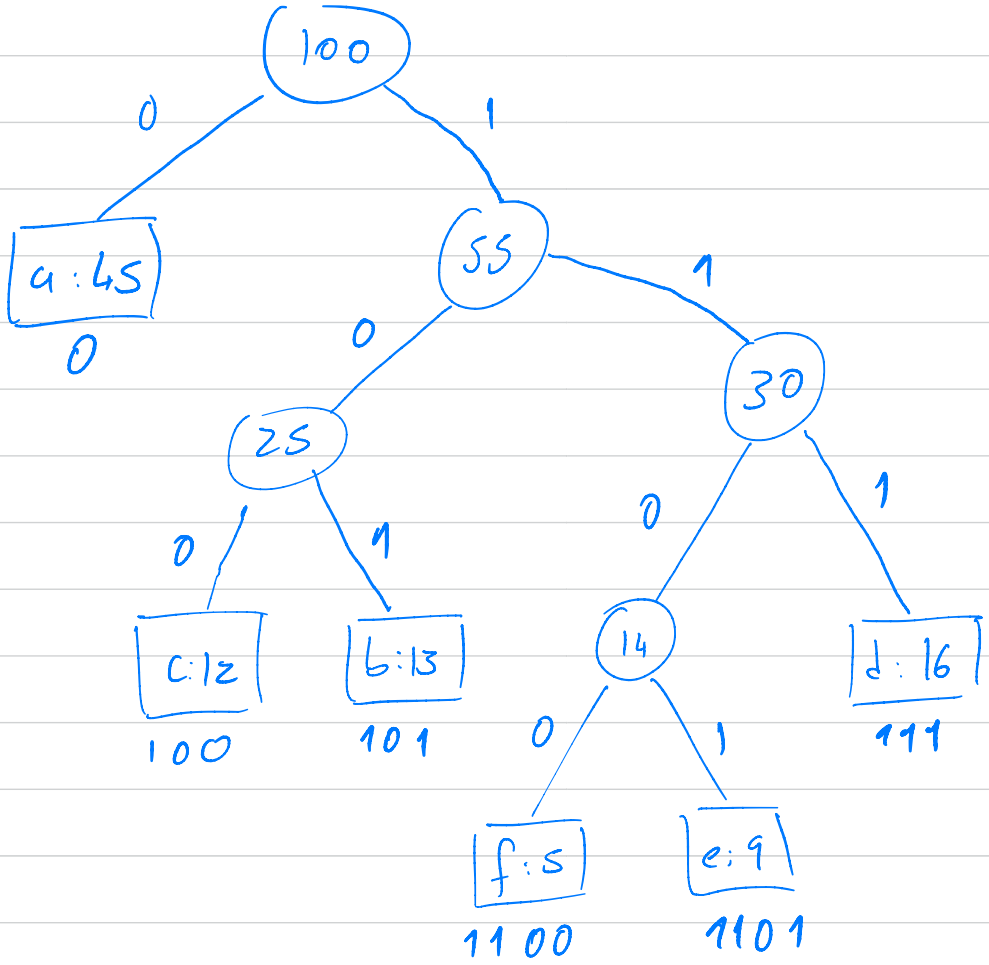
Complexidade: $O(n \cdot \log n)$
com $n = |C|$

# Algoritmo de Huffman - Exemplo

- f: 5, e: 9, c: 12, b: 13, d: 16, a: 45

# Algoritmo de Huffman - Exemplo

- f: 5, e: 9, c: 12, b: 13, d: 16, a: 45

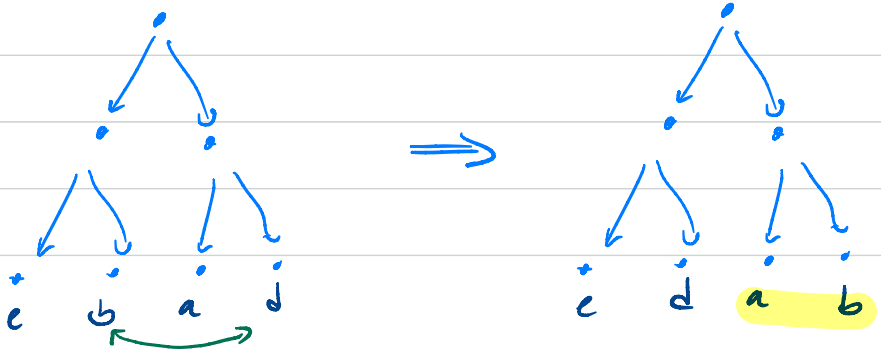


# Algoritmo de Huffman - Escolha ótima

**Lema [Algoritmo de Huffman - Escolha Ótima]**
Seja $C$ um alfabeto c/ função de frequências $f$ e $a$ e $b$ os elementos de $C$ com menor frequência; então, existe uma árvore binária ótima para $C$ e $f$ na qual $a$ e $b$ são irmãos.

**Prova**

- Seja $T$ uma árvore binária ótima para $(C, f)$. Se $a$ e $b$ são irmãos em $T$ não há nada a provar.
  Suponhamos então que $a$ e $b$ não são irmãos em $T$. Seja $T'$ obtida trocando o irmão de $a$ por $b$ em $T$.




- Resta provar q

$$B(T') \leq B(T)$$

# Algoritmo de Huffman - Escolha ótima

**Lema [Algoritmo de Huffman - Escolha Ótima]**
Seja $C$ um alfabeto c/ função de frequências $f$ e $a$ e $b$ os elementos de $C$ com menor frequência; então, existe uma árvore binária ótima para $C$ e $f$ na qual $a$ e $b$ são irmãos.

**Prova**

$$B(T) - B(T') = \sum_{c \in C} f(c) \cdot d_T(c) - \sum_{c \in C} f(c) \cdot d_{T'}(c)$$

$$= f(b) \cdot d_T(b) + f(d) \cdot d_T(d) - f(b) \cdot d_{T'}(b) - f(d) \cdot d_{T'}(d)$$

$$= f(b) \cdot \left( d_T(b) - d_{T'}(b) \right) + f(d) \cdot \left( d_T(d) - d_{T'}(d) \right)$$

$$= f(b) \left( d_T(b) - d_T(d) \right) + f(d) \left( d_T(d) - d_T(b) \right)$$

$$= \underbrace{\left( f(d) - f(b) \right)}_{\geq 0} \underbrace{\left( d_T(d) - d_T(b) \right)}_{\geq 0} \geq 0$$

**Teorema III.2** [ Huffman - Correcção ]

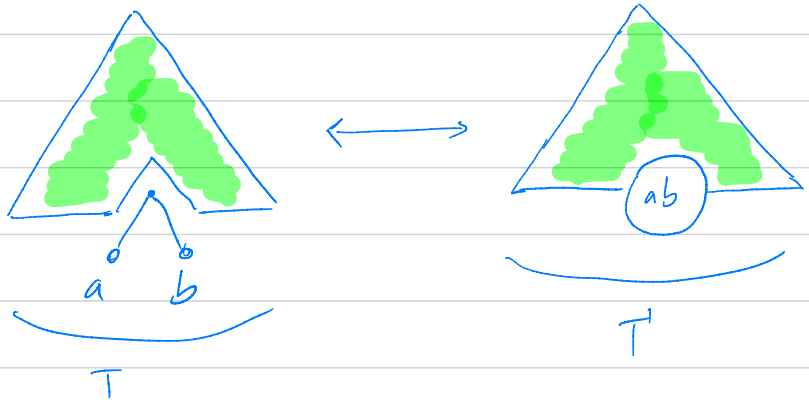
Seja $C$ um alfabeto com função de frequências $f$, o algoritmo de Huffman calcula um código binário óptimo para $C$.

**Prova**

* Dado $C$ com função de frequências $f$, provemos que o algoritmo de Huffman calcula uma árvore binária óptima para $C$ por indução no tamanho de $C$.

- **Base** $|C|=2$ $\quad C=\{a,b\}$  $\quad B(T) = f(a)+f(b)$

- **Passo**, $|C| = n = n'+1$
  - Sejam $a$ e $b$ os elementos de $C$ com menor frequência.
  - O lema III.1 garante que há uma árvore óptima na qual $a$ e $b$ são irmãos.
    Então só temos de olhar para as árvores em que $a$ e $b$ são irmãos:

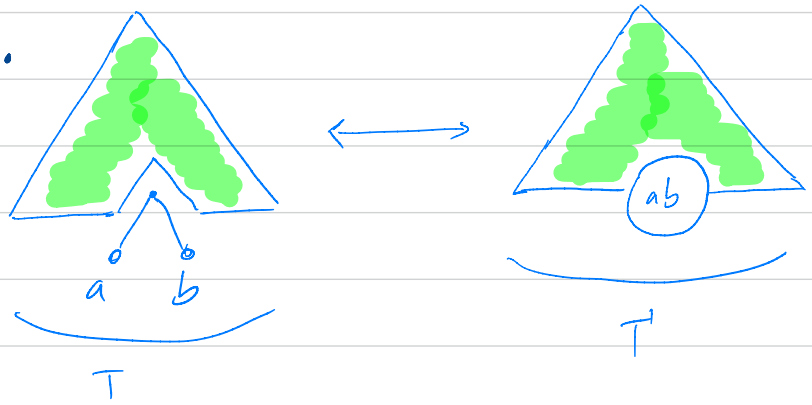



- Construímos uma nova instância instância do problema de tamanho $n-1$.

  - $C' = (C \setminus \{a,b\}) \cup \{x_{ab}\}$

  - $f' = f|_{C\setminus\{a,b\}}\,[x_{ab} \mapsto f(a)+f(b)]$

- A hipótese de indução garante que o algoritmo de Huffman calcula a árvore binária óptima para o alfabeto $C'$ com frequências dadas por $f'$!




Falta apenas argumentar que a melhor solução para $(C, f)$ corresponde necessariamente à melhor solução para $(C', f')$.

$$B(T) - B(T') = \sum_{c \in C} f(c) \cdot d_T(c) - \sum_{c \in C'} f'(c) \cdot d_{T'}(c)$$

$$= f(a) \cdot d_T(a) + f(b) \cdot d_T(b) - f'(x_{ab}) \cdot d_{T'}(x_{ab})$$

$$= (d+1)(f(a) + f(b)) - d(f(a) + f(b))$$

$$= f(a) + f(b)$$

$k \Rightarrow$ constante

$B(T') = B(T) - k$

↳ A melhor árvore para $C'$ corresponde à melhor árvore para $C$.