MC458 — Projeto e Análise de Algoritmos I

C.C. de Souza C.N. da Silva O. Lee

Antes de mais nada...

- Uma versão anterior deste conjunto de slides foi preparada por Cid Carvalho de Souza e Cândida Nunes da Silva para uma instância anterior desta disciplina.
- O que vocês tem em mãos é uma versão modificada preparada para atender a meus gostos.
- Nunca é demais enfatizar que o material é apenas um guia e não deve ser usado como única fonte de estudo. Para isso consultem a bibliografia (em especial o CLR ou CLRS).

Orlando Lee

Agradecimentos (Cid e Cândida)

- Várias pessoas contribuíram direta ou indiretamente com a preparação deste material.
- Algumas destas pessoas cederam gentilmente seus arquivos digitais enquanto outras cederam gentilmente o seu tempo fazendo correções e dando sugestões.
- Uma lista destes "colaboradores" (em ordem alfabética) é dada abaixo:
  - Célia Picinin de Mello
  - José Coelho de Pina
  - Orlando Lee
  - Paulo Feofiloff
  - Pedro Rezende
  - Ricardo Dahab
  - Zanoni Dias

Códigos de Huffman

Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.

Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.
- Reduções no tamanho dos arquivos dependem das características dos dados contidos nos mesmos. Valores típicos oscilam entre 20 e 90%.

Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.
- Reduções no tamanho dos arquivos dependem das características dos dados contidos nos mesmos. Valores típicos oscilam entre 20 e 90%.
- Exemplo: arquivo texto contendo 100.000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As *freqüências* de cada caracter no arquivo são indicadas na tabela abaixo.

Caracteres	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5

Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.
- Reduções no tamanho dos arquivos dependem das características dos dados contidos nos mesmos. Valores típicos oscilam entre 20 e 90%.
- Exemplo: arquivo texto contendo 100.000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As *freqüências* de cada caracter no arquivo são indicadas na tabela abaixo.

Caracteres	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5

- Codificação do arquivo: representar cada caracter por uma seqüência de *bits*

Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.
- Reduções no tamanho dos arquivos dependem das características dos dados contidos nos mesmos. Valores típicos oscilam entre 20 e 90%.
- Exemplo: arquivo texto contendo 100.000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As *freqüências* de cada caracter no arquivo são indicadas na tabela abaixo.

Caracteres	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5

- Codificação do arquivo: representar cada caracter por uma seqüência de *bits*
- Alternativas:

Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.
- Reduções no tamanho dos arquivos dependem das características dos dados contidos nos mesmos. Valores típicos oscilam entre 20 e 90%.
- Exemplo: arquivo texto contendo 100.000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As *freqüências* de cada caracter no arquivo são indicadas na tabela abaixo.

Caracteres	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5

- Codificação do arquivo: representar cada caracter por uma seqüência de *bits*
- Alternativas:
  1. seqüências de tamanho fixo.

Códigos de Huffman

- Códigos de Huffman: técnica de compressão de dados.
- Reduções no tamanho dos arquivos dependem das características dos dados contidos nos mesmos. Valores típicos oscilam entre 20 e 90%.
- Exemplo: arquivo texto contendo 100.000 caracteres no alfabeto $\Sigma = \{a, b, c, d, e, f\}$. As *freqüências* de cada caracter no arquivo são indicadas na tabela abaixo.

Caracteres	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5

- Codificação do arquivo: representar cada caracter por uma seqüência de *bits*
- Alternativas:
  1. seqüências de tamanho fixo.
  2. seqüências de tamanho variável.

Códigos de Huffman

Caracteres	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

Códigos de Huffman

Caracteres	*a*	*b*	*c*	*d*	*e*	*f*
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

- Qual o tamanho (em *bits*) do arquivo codificado usando os códigos acima?

Códigos de Huffman

Caracteres	*a*	*b*	*c*	*d*	*e*	*f*
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

- Qual o tamanho (em *bits*) do arquivo codificado usando os códigos acima?
- Códigos de tamanho fixo: $3 \times 100.000 = 300.000$

Códigos de Huffman

Caracteres	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101
Código de tamanho variável	0	101	100	111	1101	1100

- Qual o tamanho (em *bits*) do arquivo codificado usando os códigos acima?
- Códigos de tamanho fixo: $3 \times 100.000 = 300.000$
  Códigos de tamanho variável:

$$(\underbrace{45 \times 1}_{a} + \underbrace{13 \times 3}_{b} + \underbrace{12 \times 3}_{c} + \underbrace{16 \times 3}_{d} + \underbrace{9 \times 4}_{e} + \underbrace{5 \times 4}_{f}) \times 1.000 = 224.000$$

Ganho de $\approx 25\%$ em relação à solução anterior.

Códigos de Huffman

Problema da Codificação:

Dadas as freqüências de ocorrência dos caracteres de um arquivo, encontrar seqüências de *bits* (códigos) para representá-los de modo que o arquivo codificado tenha tamanho mínimo.

Problema da Codificação:

Dadas as freqüências de ocorrência dos caracteres de um arquivo, encontrar seqüências de *bits* (códigos) para representá-los de modo que o arquivo codificado tenha tamanho mínimo.

Definição:

Códigos livres de prefixo são aqueles onde, dados dois caracteres quaisquer i e j representados pela codificação, a seqüência de *bits* associada a i **não** é um *prefixo* da seqüência associada a j.

Códigos de Huffman

Problema da Codificação:

Dadas as freqüências de ocorrência dos caracteres de um arquivo, encontrar seqüências de *bits* (códigos) para representá-los de modo que o arquivo codificado tenha tamanho mínimo.

Definição:

Códigos livres de prefixo são aqueles onde, dados dois caracteres quaisquer i e j representados pela codificação, a seqüência de *bits* associada a i **não** é um *prefixo* da seqüência associada a j.

Importante:

Pode-se provar que sempre **existe** uma solução ótima do problema da codificação que é dado por um código *livre de prefixo*.

O processo de codificação, i.e, de geração do arquivo codificado é sempre fácil pois reduz-se a concatenar os códigos dos caracteres presentes no arquivo original em seqüência.

O processo de codificação, i.e, de geração do arquivo codificado é sempre fácil pois reduz-se a concatenar os códigos dos caracteres presentes no arquivo original em seqüência.

Exemplo: usando a codificação de tamanho variável do exemplo anterior, o arquivo original dado por *abc* seria codificado por 0101100.

Caracteres	*a*	*b*	*c*	*d*	*e*	*f*
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho variável	0	101	100	111	1101	1100

Códigos de Huffman – decodificação

Códigos de Huffman – decodificação

- A vantagem dos códigos livres de prefixo se torna evidente quando vamos decodificar o arquivo codificado.

- A vantagem dos códigos livres de prefixo se torna evidente quando vamos decodificar o arquivo codificado.
- Como nenhum código é prefixo de outro código, o código que se encontra no início do arquivo codificado não apresenta ambigüidade. Pode-se simplesmente identificar este código inicial, traduzi-lo de volta ao caracter original e repetir o processo no restante do arquivo codificado.

Códigos de Huffman – decodificação

- A vantagem dos códigos livres de prefixo se torna evidente quando vamos decodificar o arquivo codificado.
- Como nenhum código é prefixo de outro código, o código que se encontra no início do arquivo codificado não apresenta ambigüidade. Pode-se simplesmente identificar este código inicial, traduzi-lo de volta ao caracter original e repetir o processo no restante do arquivo codificado.
- Exemplo: no exemplo anterior, o arquivo codificado contendo os *bits* 001011101 divide-se de **forma unívoca** em 0 0 101 1101, ou seja, corresponde ao arquivo original dado por *aabe*.

	a	*b*	*c*	*d*	*e*	*f*
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho variável	0	101	100	111	1101	1100

Códigos de Huffman

- Como representar de maneira conveniente uma codificação livre de prefixo de modo a facilitar o processo de decodificação?

Códigos de Huffman

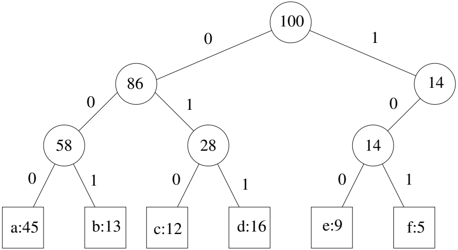
- Como representar de maneira conveniente uma codificação livre de prefixo de modo a facilitar o processo de decodificação?
- Solução: usar uma árvore binária.

- Como representar de maneira conveniente uma codificação livre de prefixo de modo a facilitar o processo de decodificação?
- Solução: usar uma árvore binária.
  O filho esquerdo está associado ao *bit* ZERO enquanto o filho direito está associado ao *bit* UM. Nas folhas encontram-se os caracteres presentes no arquivo original.

Códigos de Huffman

Vejamos como ficam as árvores que representam os códigos do exemplo anterior.

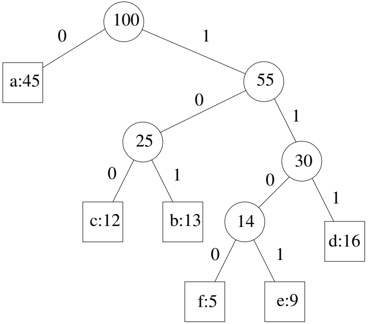
	a	*b*	*c*	*d*	*e*	*f*
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho fixo	000	001	010	011	100	101



Códigos de Huffman

Vejamos como ficam as árvores que representam os códigos do exemplo anterior.

	a	*b*	*c*	*d*	*e*	*f*
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho variável	0	101	100	111	1101	1100



- Pode-se mostrar (Exercício!) que uma codificação ótima sempre pode ser representada por uma árvore binária cheia, na qual cada nó interno tem exatamente dois filhos.

Códigos de Huffman

- Pode-se mostrar (Exercício!) que uma codificação ótima sempre pode ser representada por uma árvore binária cheia, na qual cada nó interno tem exatamente dois filhos.
- Então podemos restringir nossa atenção às árvores binárias cheias com $|C|$ folhas e $|C| - 1$ nós internos (Exercício!), onde C é o conjunto de caracteres do alfabeto no qual está escrito o arquivo original.

Códigos de Huffman

Computando o tamanho do arquivo codificado:

Se T é a árvore que representa a codificação, $d_T(c)$ é a profundidade da folha representado o caracter c e $\mathrm{fr}[c]$ é a sua freqüência, o tamanho do arquivo codificado será dado por:

$$B(T) = \sum_{c \in C} \mathrm{fr}[c] d_T(c).$$

Computando o tamanho do arquivo codificado:

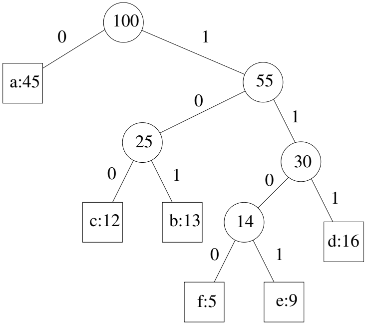
Se T é a árvore que representa a codificação, $d_T(c)$ é a profundidade da folha representado o caracter c e $\mathrm{fr}[c]$ é a sua freqüência, o tamanho do arquivo codificado será dado por:

$$B(T) = \sum_{c \in C} \mathrm{fr}[c] d_T(c).$$

Dizemos que $B(T)$ é o **custo** da árvore T.

Códigos de Huffman

	a	b	c	d	e	f
Freqüência (em milhares)	45	13	12	16	9	5
Código de tamanho variável	0	101	100	111	1101	1100



$$B(T) = \sum_{c \in C} \mathrm{fr}[c] d_T(c) = 45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4 = 224.$$

Códigos de Huffman

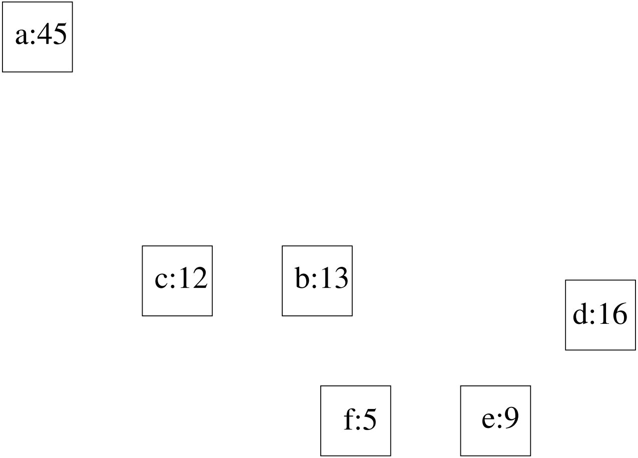
Códigos de Huffman

- **Idéia do algoritmo de Huffman**: Começar com $|C|$ nós (árvores triviais) e realizar sequencialmente $|C| - 1$ operações de "intercalação" das raízes de duas árvores.

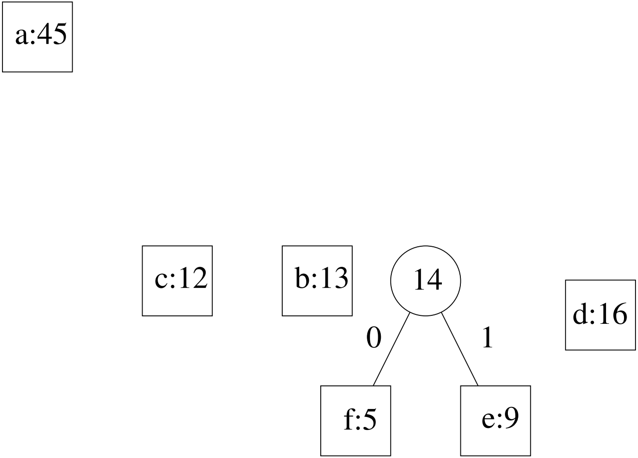
Códigos de Huffman

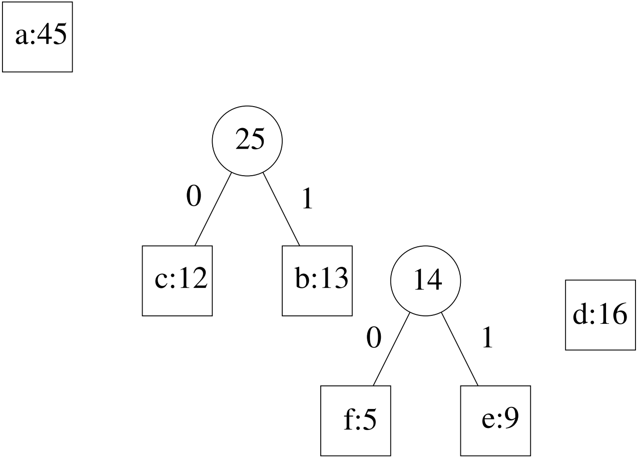
- **Idéia do algoritmo de Huffman**: Começar com $|C|$ nós (árvores triviais) e realizar sequencialmente $|C| - 1$ operações de "intercalação" das raízes de duas árvores.
- Cada uma destas intercalações dá origem a um novo nó interno, que será o pai dos nós que participaram da intercalação (raiz da nova árvore).

- **Idéia do algoritmo de Huffman**: Começar com $|C|$ nós (árvores triviais) e realizar sequencialmente $|C| - 1$ operações de "intercalação" das raízes de duas árvores.
- Cada uma destas intercalações dá origem a um novo nó interno, que será o pai dos nós que participaram da intercalação (raiz da nova árvore).
- A escolha do par de nós (raízes) que dará origem à intercalação em cada passo depende da soma das freqüências das folhas das subárvores com raízes nos nós que ainda não participaram de intercalações.

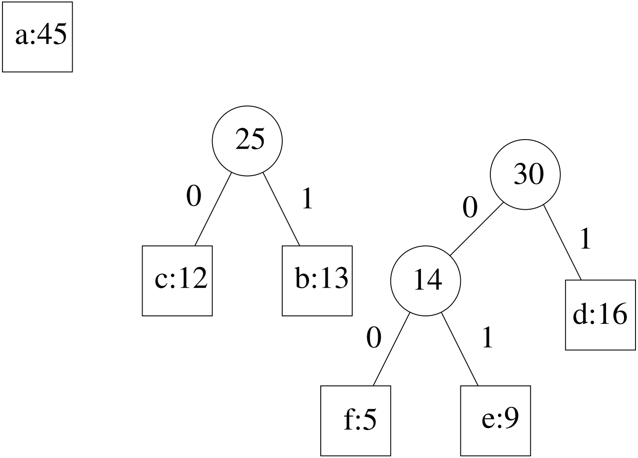

a:45
c:12
b:13
d:16
f:5
e:9

Algoritmo de Huffman

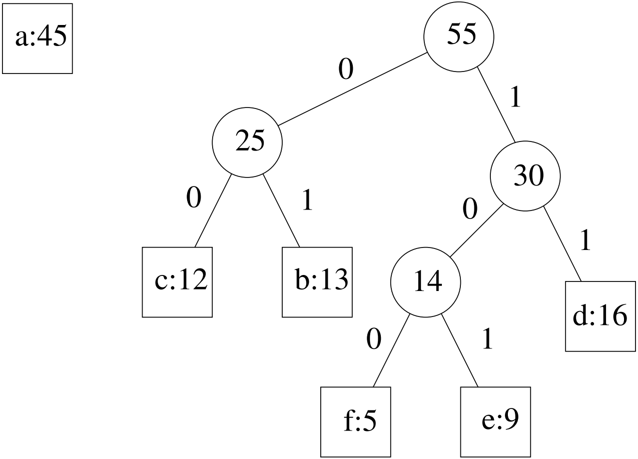



a:45
25
0
1
c:12
b:13
14
0
1
d:16
f:5
e:9

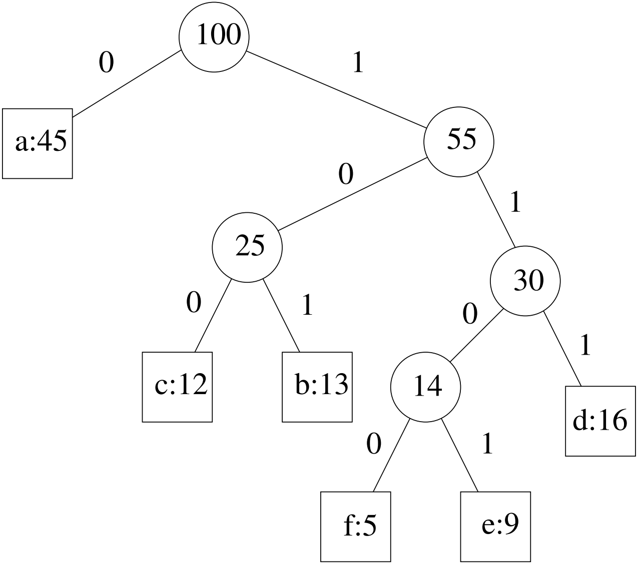
Algoritmo de Huffman



Algoritmo de Huffman



Algoritmo de Huffman



Algoritmo de Huffman

$\textsc{Huffman}(C, \mathrm{fr})$

$n \leftarrow |C|$;
$Q \leftarrow C$ $\quad \triangleright$ Q é uma fila de prioridades com chave fr
para $i \leftarrow 1$ **até** $n-1$ **faça**
    alocar novo registro z $\quad \triangleright$ nó de T
    $z.esq \leftarrow x \leftarrow \textsc{Extract-Min}(Q)$
    $z.dir \leftarrow y \leftarrow \textsc{Extract-Min}(Q)$
    $z.\mathrm{fr} \leftarrow x.\mathrm{fr} + y.\mathrm{fr}$
    $\textsc{Insert}(Q, z)$
devolva $\textsc{Extract-Min}(Q)$ $\quad \triangleright$ devolve a raiz da árvore

$\textsc{Extract-Min}(Q)$ remove um elemento com menor chave de Q e devolve um apontador para este.

Algoritmo de Huffman

Algoritmo de Huffman

- O processo de intercalar duas folhas x e y pode ser visto como a substituição dos caracteres x e y por um novo caractere z com frequência $\mathrm{fr}[z] = \mathrm{fr}[x] + \mathrm{fr}[y]$. As frequências dos demais caracteres não são modificadas.

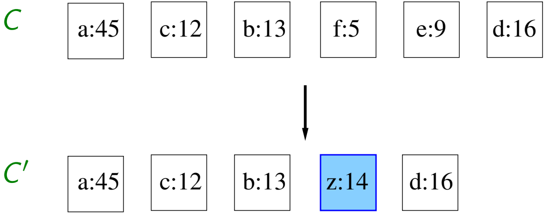
Algoritmo de Huffman

- O processo de intercalar duas folhas x e y pode ser visto como a substituição dos caracteres x e y por um novo caractere z com frequência $\mathrm{fr}[z] = \mathrm{fr}[x] + \mathrm{fr}[y]$. As frequências dos demais caracteres não são modificadas.
- Assim, o algoritmo pode ser visto como um **método recursivo** que identifica dois caracteres com as menores frequência e recursivamente calcula uma **árvore ótima** T' do novo alfabeto $C' = C - \{x, y\} \cup \{z\}$.

Algoritmo de Huffman

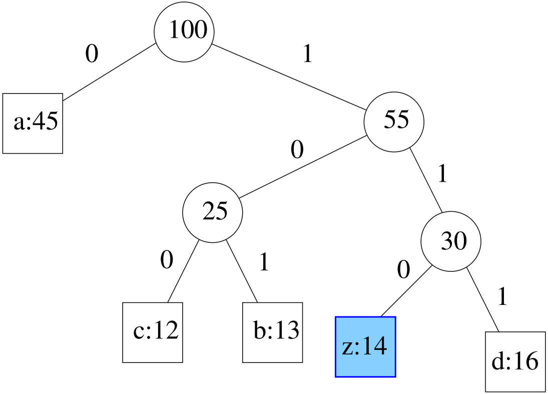
- O processo de intercalar duas folhas x e y pode ser visto como a substituição dos caracteres x e y por um novo caractere z com frequência $\mathrm{fr}[z] = \mathrm{fr}[x] + \mathrm{fr}[y]$. As frequências dos demais caracteres não são modificadas.
- Assim, o algoritmo pode ser visto como um **método recursivo** que identifica dois caracteres com as menores frequência e recursivamente calcula uma **árvore ótima** T' do novo alfabeto $C' = C - \{x, y\} \cup \{z\}$.
- Uma **árvore ótima** T do alfabeto original C é obtida a partir de T' inserindo x e y como sendo filhos de z (que torna-se um nó interno).

Algoritmo de Huffman

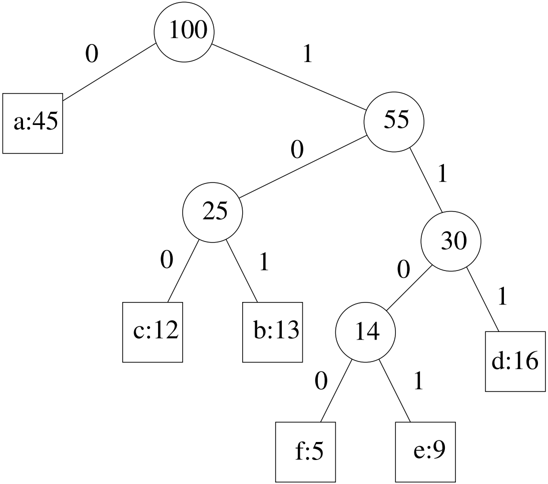


$C' = C - \{e, f\} \cup \{z\}$ e $\mathrm{fr}[z] = \mathrm{fr}[e] + \mathrm{fr}[f]$.

Algoritmo de Huffman



Árvore ótima T' para C'.



Árvore (ótima) T obtida de T'.

Corretude do algoritmo de Huffman

Lema 1: (escolha gulosa)

Seja C um alfabeto onde cada caractere $c \in C$ tem freqüência $\mathrm{fr}[c]$. Sejam x e y dois caracteres em C com as **menores** freqüências. Então, existe uma árvore binária que representa um código ótimo livre de prefixo para C na qual x e y são folhas irmãs.

Lema 1: (escolha gulosa)

Seja C um alfabeto onde cada caractere $c \in C$ tem freqüência $\mathrm{fr}[c]$. Sejam x e y dois caracteres em C com as **menores** freqüências. Então, existe uma árvore binária que representa um código ótimo livre de prefixo para C na qual x e y são folhas irmãs.

Prova do Lema 1:

Lema 1: (escolha gulosa)

Seja C um alfabeto onde cada caractere $c \in C$ tem freqüência $\mathrm{fr}[c]$. Sejam x e y dois caracteres em C com as **menores** freqüências. Então, existe uma árvore binária que representa um código ótimo livre de prefixo para C na qual x e y são folhas irmãs.

Prova do Lema 1:

- Seja T uma **árvore ótima** para C.

Lema 1: (escolha gulosa)

Seja C um alfabeto onde cada caractere $c \in C$ tem freqüência $\mathrm{fr}[c]$. Sejam x e y dois caracteres em C com as **menores** freqüências. Então, existe uma árvore binária que representa um código ótimo livre de prefixo para C na qual x e y são folhas irmãs.

Prova do Lema 1:

- Seja T uma **árvore ótima** para C.
- Sejam a e b duas folhas irmãs mais profundas de T e sejam x e y as folhas de T de menor freqüência.

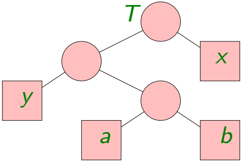
Lema 1: (escolha gulosa)

Seja C um alfabeto onde cada caractere $c \in C$ tem freqüência $\mathrm{fr}[c]$. Sejam x e y dois caracteres em C com as **menores** freqüências. Então, existe uma árvore binária que representa um código ótimo livre de prefixo para C na qual x e y são folhas irmãs.

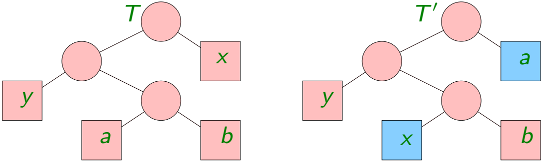
Prova do Lema 1:

- Seja T uma **árvore ótima** para C.
- Sejam a e b duas folhas irmãs mais profundas de T e sejam x e y as folhas de T de menor freqüência.
- **Idéia:** a partir de T, obter uma outra **árvore ótima** T' na qual x e y são duas folhas irmãs.

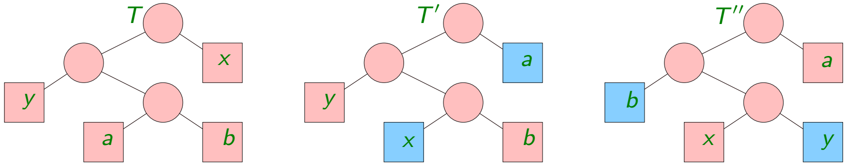
Corretude do algoritmo de Huffman

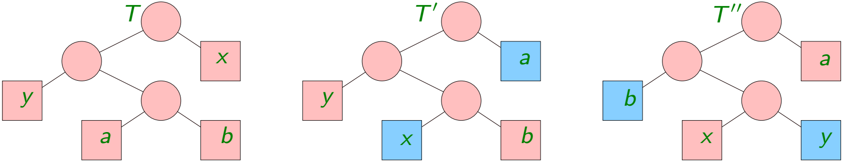


Corretude do algoritmo de Huffman

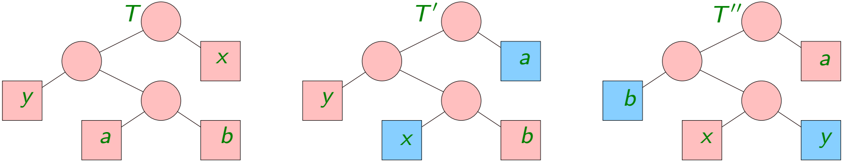


Corretude do algoritmo de Huffman





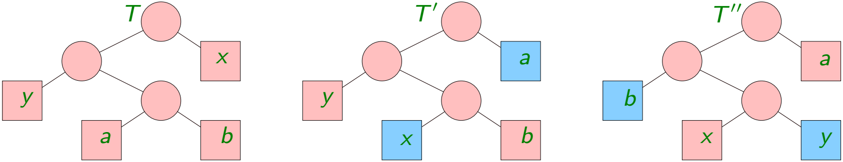
$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} \mathrm{fr}[c] d_T(c) - \sum_{c \in C} \mathrm{fr}[c] d_{T'}(c) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_{T'}(x) - \mathrm{fr}[a] d_{T'}(a) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_T(a) - \mathrm{fr}[a] d_T(x) \\
&= (\mathrm{fr}[a] - \mathrm{fr}[x])(d_T(a) - d_T(x)) \geq 0
\end{aligned}
$$



$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} \mathrm{fr}[c] d_T(c) - \sum_{c \in C} \mathrm{fr}[c] d_{T'}(c) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_{T'}(x) - \mathrm{fr}[a] d_{T'}(a) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_T(a) - \mathrm{fr}[a] d_T(x) \\
&= (\mathrm{fr}[a] - \mathrm{fr}[x])(d_T(a) - d_T(x)) \geq 0
\end{aligned}
$$

Assim, $B(T) \geq B(T')$.

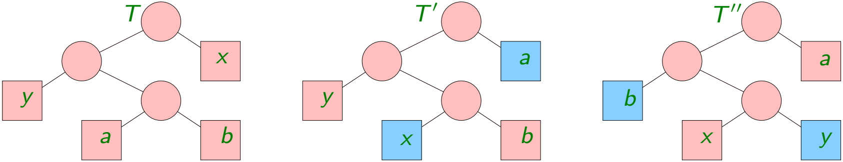
Corretude do algoritmo de Huffman



$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} \mathrm{fr}[c] d_T(c) - \sum_{c \in C} \mathrm{fr}[c] d_{T'}(c) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_{T'}(x) - \mathrm{fr}[a] d_{T'}(a) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_T(a) - \mathrm{fr}[a] d_T(x) \\
&= (\mathrm{fr}[a] - \mathrm{fr}[x])(d_T(a) - d_T(x)) \geq 0
\end{aligned}
$$

Assim, $B(T) \geq B(T')$.

Analogamente $B(T') \geq B(T'')$.



$$
\begin{aligned}
B(T) - B(T') &= \sum_{c \in C} \mathrm{fr}[c] d_T(c) - \sum_{c \in C} \mathrm{fr}[c] d_{T'}(c) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_{T'}(x) - \mathrm{fr}[a] d_{T'}(a) \\
&= \mathrm{fr}[x] d_T(x) + \mathrm{fr}[a] d_T(a) - \mathrm{fr}[x] d_T(a) - \mathrm{fr}[a] d_T(x) \\
&= (\mathrm{fr}[a] - \mathrm{fr}[x])(d_T(a) - d_T(x)) \geq 0
\end{aligned}
$$

Assim, $B(T) \geq B(T')$.

Analogamente $B(T') \geq B(T'')$.

Como T é ótima, T'' é ótima e o resultado vale.

Corretude do algoritmo de Huffman

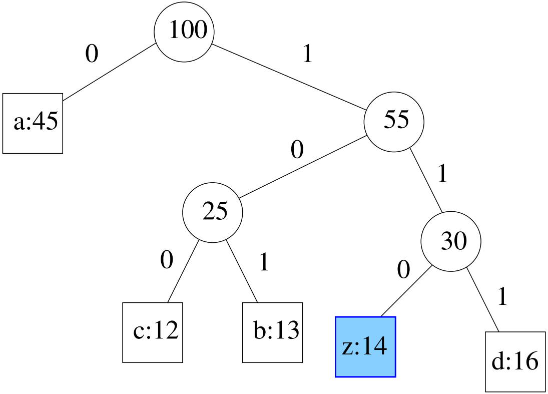
Lema 2: (subestrutura ótima)

Seja C um alfabeto com freqüência $\mathrm{fr}[c]$ definida para cada caractere $c \in C$. Sejam x e y dois caracteres de C com as menores freqüências. Seja C' o alfabeto obtido pela remoção de x e y e pela inclusão de um novo caractere z, ou seja, $C' = C - \{x, y\} \cup \{z\}$. As freqüências dos caracteres em $C' \cap C$ são as mesmas que em C e $\mathrm{fr}[z]$ é definida como sendo $\mathrm{fr}[z] = \mathrm{fr}[x] + \mathrm{fr}[y]$.
Seja T' uma árvore binária que representa um código ótimo livre de prefixo para C'. Então a árvore binária T, obtida de T' substituindo-se o nó (folha) z por um nó interno tendo x e y como filhos, representa um código ótimo livre de prefixo para C.

C	a:45	c:12	b:13	f:5	e:9	d:16
C'	a:45	c:12	b:13	z:14	d:16	

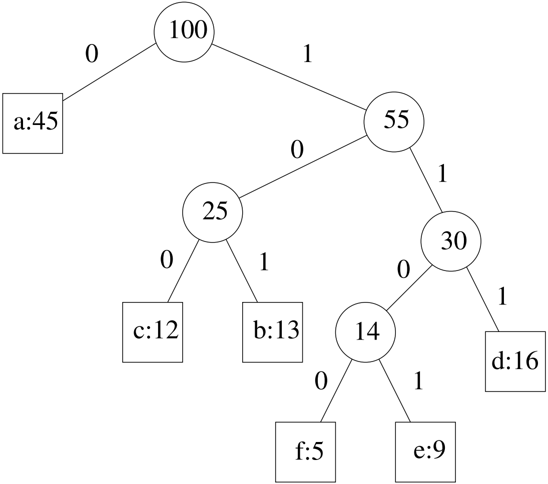
$$C' = C - \{e, f\} \cup \{z\} \text{ e } \mathrm{fr}[z] = \mathrm{fr}[e] + \mathrm{fr}[f].$$

Algoritmo de Huffman



Árvore ótima T' para C'.

Algoritmo de Huffman



Árvore (ótima) T obtida de T'.

Corretude do algoritmo de Huffman

Prova do Lema 2:

Prova do Lema 2:

- Comparando os custos de T e T':

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c]d_{T'}(c) = \mathrm{fr}[c]d_T(c)$.

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c]d_{T'}(c) = \mathrm{fr}[c]d_T(c)$.
  - $\mathrm{fr}[x]d_T(x) + \mathrm{fr}[y]d_T(y) = (\mathrm{fr}[x] + \mathrm{fr}[y])(d_{T'}(z) + 1) = \mathrm{fr}[z]d_{T'}(z) + (\mathrm{fr}[x] + \mathrm{fr}[y])$.

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c]d_{T'}(c) = \mathrm{fr}[c]d_T(c)$.
  - $\mathrm{fr}[x]d_T(x) + \mathrm{fr}[y]d_T(y) = (\mathrm{fr}[x] + \mathrm{fr}[y])(d_{T'}(z) + 1) = \mathrm{fr}[z]d_{T'}(z) + (\mathrm{fr}[x] + \mathrm{fr}[y])$.
- Logo, $B(T') = B(T) - \mathrm{fr}[x] - \mathrm{fr}[y]$.

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c]d_{T'}(c) = \mathrm{fr}[c]d_T(c)$.
  - $\mathrm{fr}[x]d_T(x) + \mathrm{fr}[y]d_T(y) = (\mathrm{fr}[x] + \mathrm{fr}[y])(d_{T'}(z) + 1) = \mathrm{fr}[z]d_{T'}(z) + (\mathrm{fr}[x] + \mathrm{fr}[y])$.
- Logo, $B(T') = B(T) - \mathrm{fr}[x] - \mathrm{fr}[y]$.
- Por contradição, suponha que exista T'' tal que $B(T'') < B(T)$.

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c]d_{T'}(c) = \mathrm{fr}[c]d_T(c)$.
  - $\mathrm{fr}[x]d_T(x) + \mathrm{fr}[y]d_T(y) = (\mathrm{fr}[x] + \mathrm{fr}[y])(d_{T'}(z) + 1) = \mathrm{fr}[z]d_{T'}(z) + (\mathrm{fr}[x] + \mathrm{fr}[y])$.
- Logo, $B(T') = B(T) - \mathrm{fr}[x] - \mathrm{fr}[y]$.
- Por contradição, suponha que exista T'' tal que $B(T'') < B(T)$. Pelo Lema 1, podemos supor que x e y são folhas irmãs em T''.

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c]d_{T'}(c) = \mathrm{fr}[c]d_T(c)$.
  - $\mathrm{fr}[x]d_T(x) + \mathrm{fr}[y]d_T(y) = (\mathrm{fr}[x] + \mathrm{fr}[y])(d_{T'}(z) + 1) = \mathrm{fr}[z]d_{T'}(z) + (\mathrm{fr}[x] + \mathrm{fr}[y])$.
- Logo, $B(T') = B(T) - \mathrm{fr}[x] - \mathrm{fr}[y]$.
- Por contradição, suponha que exista T'' tal que $B(T'') < B(T)$. Pelo Lema 1, podemos supor que x e y são folhas irmãs em T''. Seja T''' a árvore obtida de T'' pela substituição de x e y por uma folha z com freqüência $\mathrm{fr}[z] = \mathrm{fr}[x] + \mathrm{fr}[y]$. O custo de T''' é

$$B(T''') = B(T'') - \mathrm{fr}[x] - \mathrm{fr}[y]$$

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c] d_{T'}(c) = \mathrm{fr}[c] d_T(c)$.
  - $\mathrm{fr}[x] d_T(x) + \mathrm{fr}[y] d_T(y) = (\mathrm{fr}[x] + \mathrm{fr}[y])(d_{T'}(z) + 1) = \mathrm{fr}[z] d_{T'}(z) + (\mathrm{fr}[x] + \mathrm{fr}[y])$.
- Logo, $B(T') = B(T) - \mathrm{fr}[x] - \mathrm{fr}[y]$.
- Por contradição, suponha que exista T'' tal que $B(T'') < B(T)$. Pelo Lema 1, podemos supor que x e y são folhas irmãs em T''. Seja T''' a árvore obtida de T'' pela substituição de x e y por uma folha z com freqüência $\mathrm{fr}[z] = \mathrm{fr}[x] + \mathrm{fr}[y]$. O custo de T''' é

$$B(T''') = B(T'') - \mathrm{fr}[x] - \mathrm{fr}[y] < B(T) - \mathrm{fr}[x] - \mathrm{fr}[y]$$

Prova do Lema 2:

- Comparando os custos de T e T':
  - Se $c \in C - \{x, y\}$, então $\mathrm{fr}[c]d_{T'}(c) = \mathrm{fr}[c]d_T(c)$.
  - $\mathrm{fr}[x]d_T(x) + \mathrm{fr}[y]d_T(y) = (\mathrm{fr}[x] + \mathrm{fr}[y])(d_{T'}(z) + 1) = \mathrm{fr}[z]d_{T'}(z) + (\mathrm{fr}[x] + \mathrm{fr}[y])$.
- Logo, $B(T') = B(T) - \mathrm{fr}[x] - \mathrm{fr}[y]$.
- Por contradição, suponha que exista T'' tal que $B(T'') < B(T)$. Pelo Lema 1, podemos supor que x e y são folhas irmãs em T''. Seja T''' a árvore obtida de T'' pela substituição de x e y por uma folha z com freqüência $\mathrm{fr}[z] = \mathrm{fr}[x] + \mathrm{fr}[y]$. O custo de T''' é

$$B(T''') = B(T'') - \mathrm{fr}[x] - \mathrm{fr}[y] < B(T) - \mathrm{fr}[x] - \mathrm{fr}[y] = B(T'),$$

contrariando o fato de T' ser uma árvore ótima para C'.

Corretude do algoritmo de Huffman

Corretude do algoritmo de Huffman

Teorema:

HUFFMAN constrói um código ótimo (livre de prefixo) para C, f.

Corretude do algoritmo de Huffman

Teorema:

HUFFMAN constrói um código ótimo (livre de prefixo) para C, f.

Segue imediatamente dos Lemas 1 e 2.

Passos do projeto de algoritmos gulosos: resumo

Passos do projeto de algoritmos gulosos: resumo

1. Formule o problema como um **problema de otimização** no qual uma escolha é feita, restando-nos então resolver um único subproblema a resolver.

Passos do projeto de algoritmos gulosos: resumo

1. Formule o problema como um **problema de otimização** no qual uma escolha é feita, restando-nos então resolver um único subproblema a resolver.
2. Provar que existe uma solução ótima do problema que atende à **escolha gulosa**, ou seja, que faz a mesma escolha.

Passos do projeto de algoritmos gulosos: resumo

1. Formule o problema como um **problema de otimização** no qual uma escolha é feita, restando-nos então resolver um único subproblema a resolver.
2. Provar que existe uma solução ótima do problema que atende à **escolha gulosa**, ou seja, que faz a mesma escolha.
3. Demonstrar que, uma vez feita a escolha gulosa, o que resta a resolver é um subproblema tal que se combinarmos a resposta ótima deste subproblema com o(s) elemento(s) da escolha gulosa, chega-se à solução ótima do problema original.

Passos do projeto de algoritmos gulosos: resumo

1. Formule o problema como um **problema de otimização** no qual uma escolha é feita, restando-nos então resolver um único subproblema a resolver.
2. Provar que existe uma solução ótima do problema que atende à **escolha gulosa**, ou seja, que faz a mesma escolha.
3. Demonstrar que, uma vez feita a escolha gulosa, o que resta a resolver é um subproblema tal que se combinarmos a resposta ótima deste subproblema com o(s) elemento(s) da escolha gulosa, chega-se à solução ótima do problema original.
   Esta é a parte que requer mais engenhosidade!

Passos do projeto de algoritmos gulosos: resumo

1. Formule o problema como um **problema de otimização** no qual uma escolha é feita, restando-nos então resolver um único subproblema a resolver.
2. Provar que existe uma solução ótima do problema que atende à **escolha gulosa**, ou seja, que faz a mesma escolha.
3. Demonstrar que, uma vez feita a escolha gulosa, o que resta a resolver é um subproblema tal que se combinarmos a resposta ótima deste subproblema com o(s) elemento(s) da escolha gulosa, chega-se à solução ótima do problema original.
   Esta é a parte que requer mais engenhosidade!
   Normalmente a prova começa com uma solução ótima *genérica* e a modificamos para que inclua o(s) elemento(s) identificados pela escolha gulosa.