"Equipado com seus 5 sentidos, o homem explora o universo ao seu redor e chama a aventura de ciência" (Edwin Hubble).

Códigos de Huffman

Paulo Ricardo Lisboa de Almeida

DSBD

UFPR

Códigos de Huffman

Algoritmo guloso de compressão.

- Tipicamente 20% a 90% de compressão.
- Depende da característica do dado.

Exemplo

Considere um arquivo texto com 100.000 caracteres.

Os caracteres no arquivo são: a, b, c, d, e, f.

Considerando que os caracteres são ASCII, qual o tamanho do arquivo?

Exemplo

Considere um arquivo texto com 100.000 caracteres.

Os caracteres no arquivo são: a, b, c, d, e, f.

Considerando que os caracteres são ASCII, qual o tamanho do arquivo?

tamanho = 100.000 * 8 = 800.000 bits = 100.000 Bytes = 98KB

Exemplo

Considere um arquivo texto com 100.000 caracteres.

Os caracteres no arquivo são: a, b, c, d, e, f.

Considerando que os caracteres são ASCII, qual o tamanho do arquivo?

tamanho = 100.000 * 8 = 800.000 bits = 100.000 Bytes = 98KB

Mas como temos apenas 6 caracteres, quantos bits são necessários no mínimo para representar cada caractere?

Exemplo

Considere um arquivo texto com 100.000 caracteres.

Os caracteres no arquivo são: a, b, c, d, e, f.

Considerando que os caracteres são ASCII, qual o tamanho do arquivo?

tamanho = 100.000 * 8 = 800.000 bits = 100.000 Bytes = 98KB

Mas como temos apenas 6 caracteres, quantos bits são necessários no mínimo para representar cada caractere?

3 bits: a = 000_2; b = 001_2; c = 010_2; d = 011_2; e = 100_2; f = 101_2.

tamanho = 100.000 * 3 = 300.000 bits = 37.500 Bytes = 37KB

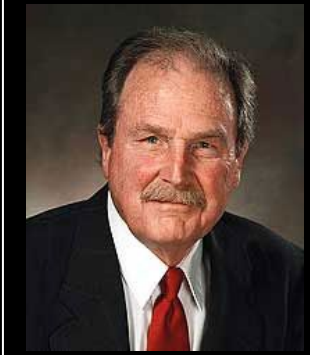
Códigos de Huffman

É possível fazer melhor com códigos de Huffman.

Primeiro, construir uma tabela de frequências (Histograma) contendo a frequência da cada caractere.

Considere que a tabela do exemplo é:

char	a	b	c	d	e	f
Frequência	45.000	13.000	12.000	16.000	9.000	5.000



David A. Huffman
09/08/1925 - 07/10/1999

Cientista da Computação.
- Código de Huffman
- Origami Computacional

www.computer.org/profiles/david-huffman

Códigos de Huffman

Ideia: Usar uma codificação de tamanho variável.

Caracteres frequentes recebem códigos menores.

char	a	b	c	d	e	f
Frequência	45.000	13.000	12.000	16.000	9.000	5.000
Código	0	101	100	111	1101	1100

Códigos de Huffman

Ideia: Usar uma codificação de tamanho variável.

Caracteres frequentes recebem códigos menores.

Tamanho = 45.000 * 1 + (13.000 + 12.000 + 16.000) * 3 + (9.000 + 5.000) * 4 = 224.000 bits
= 28.000 Bytes = 27KB = 28% do tamanho do arquivo original.

char	a	b	c	d	e	f
Frequência	45.000	13.000	12.000	16.000	9.000	5.000
Código	0	101	100	111	1101	1100

Códigos de prefixo

Em códigos de prefixo, nenhum código de palavra é também prefixo de algum outro código de palavra.

A tradução não é ambígua.

Um código de prefixo sempre consegue a **compressão ótima** de um código de caracteres.

Codificando

Para codificar, basta concatenar o código de cada caractere no arquivo.

Exemplo: face = 1100010011101

char	a	b	c	d	e	f
Frequência	45.000	13.000	12.000	16.000	9.000	5.000
Código	0	101	100	111	1101	1100

Faça você mesmo

Decodifique a palavra 100011001101.

char	a	b	c	d	e	f
Frequência	45.000	13.000	12.000	16.000	9.000	5.000
Código	0	101	100	111	1101	1100

Faça você mesmo

Decodifique a palavra 100011001101.

100011001101 = cafe

char	a	b	c	d	e	f
Frequência	45.000	13.000	12.000	16.000	9.000	5.000
Código	0	101	100	111	1101	1100

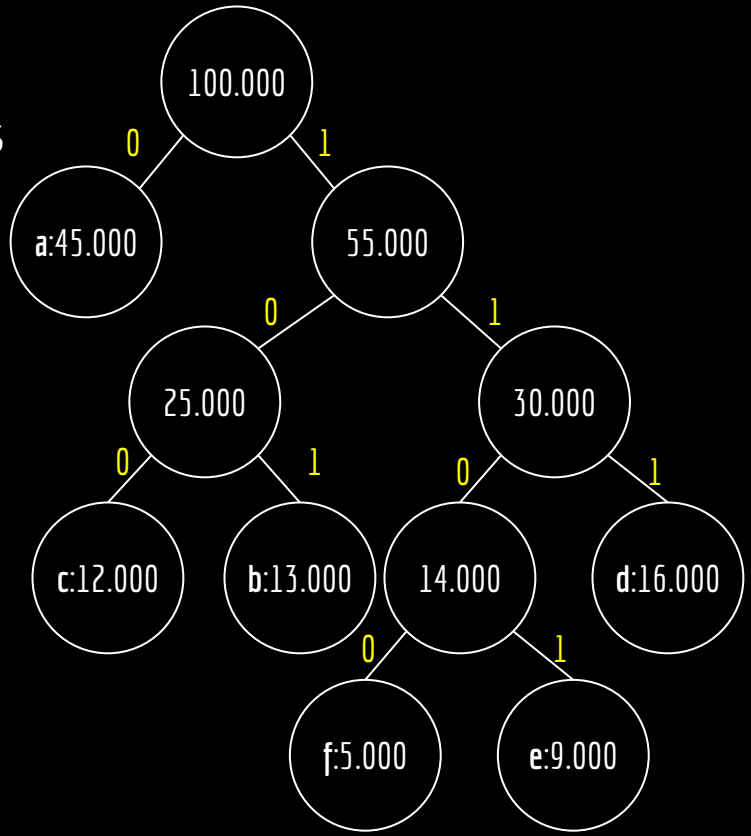
Navegando

Para decodificar, podemos usar uma árvore binária onde as folhas são os caracteres.

0 significa vá para a esquerda, e 1 significa vá para a direita.

No exemplo, a chave armazenada é o número de ocorrências das folhas combinadas.

char	a	b	c	d	e	f
Frequência	45.000	13.000	12.000	16.000	9.000	5.000
Código	0	101	100	111	1101	1100



Custo em bits

Dada uma árvore de prefixos T, o número de bits (**custo**) necessários para armazenar o arquivo é:

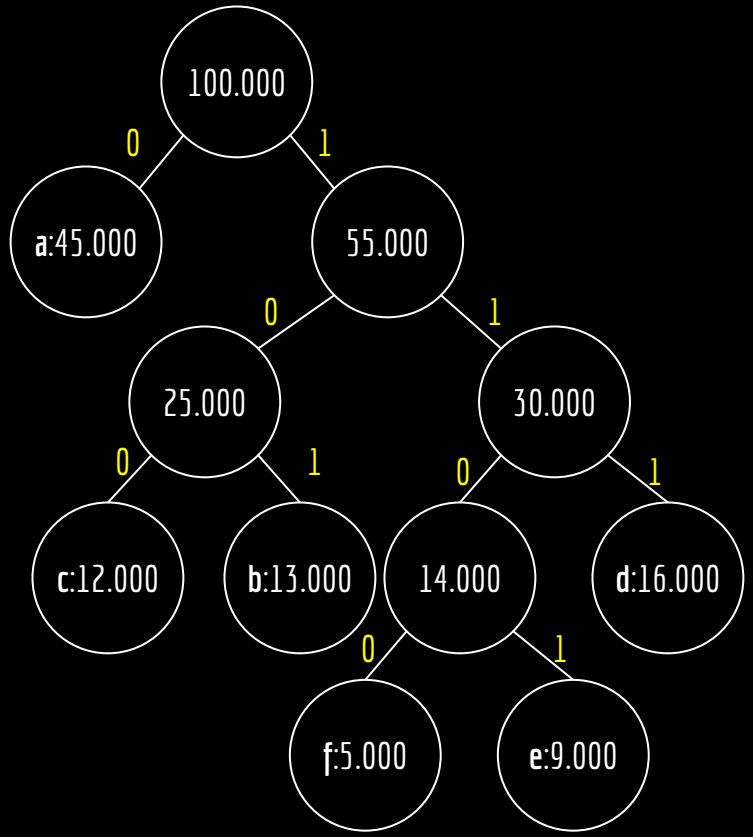
$$custo = B(T) = \sum_{c \in C} freq(c) \times d_t(c)$$

Onde *C* é o alfabeto (caracteres possíveis), *freq(c)* é a frequência que o caractere *c* aparece no texto, e $d_t(c)$ é a profundidade da folha que contém *c*.

Faça você mesmo

Calcule o custo da árvore do exemplo.

$$custo = B(T) = \sum_{c \in C} freq(c) \times d_t(c)$$

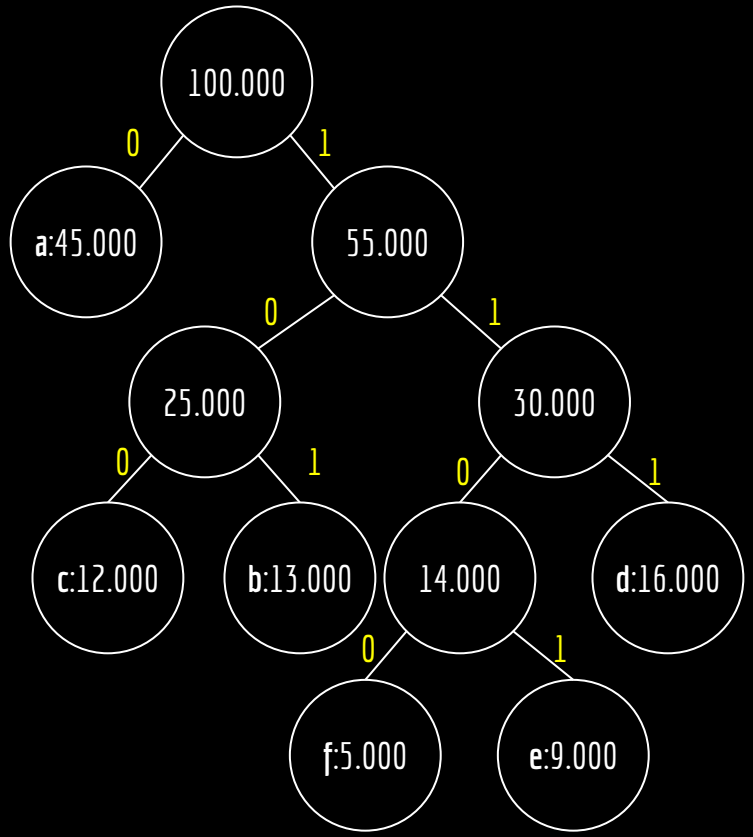


Faça você mesmo

Calcule o custo da árvore do exemplo.

$$custo = B(T) = \sum_{c \in C} freq(c) \times d_t(c)$$

custo = 224.000 bits



Algoritmo

```
função huffman(C)
entrada: histograma C contendo as frequências de cada caractere.
saída: a árvore de código de Huffman.

n = |C|
Q = fila ordenada pelas frequências em ordem não decrescente de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Palavras

abed
bade
bead
cab
bad
bed
ace
be

C

char	a	b	c	d	e
Freq.	6	7	2	5	6

huffman(C)

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Palavras

abed
bade
bead
cab
bad
bed
ace
be

C

char	a	b	c	d	e
Freq.	6	7	2	5	6

huffman(C)
n
5

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

c	d	a	e	b
2	5	6	6	7

huffman(C)
n
5

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Uma boa ideia é usar uma Min-HEAP em Q. No teste de mesa do exemplo será usada uma fila ordenada convencional para simplificar a visualização.

Teste de Mesa

Q

c	d	a	e	b
2	5	6	6	7

huffman(C)

n	i
5	1

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

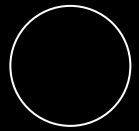
Teste de Mesa

Q

c	d	a	e	b
2	5	6	6	7

huffman(C)

n	i
5	1

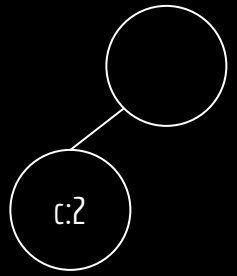


```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

d	a	e	b
5	6	6	7



huffman(C)

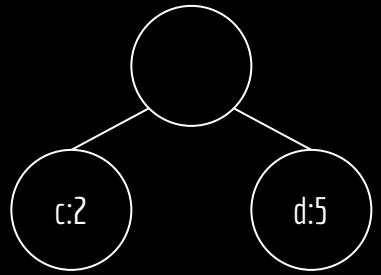
n	i
5	1

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

a	e	b
6	6	7



huffman(C)

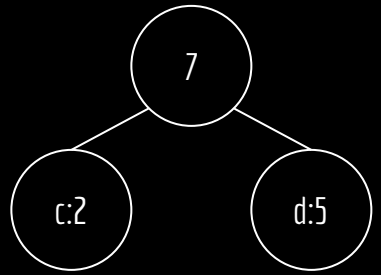
n	i
5	1

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

a	e	b
6	6	7



huffman(C)

n	i
5	1

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

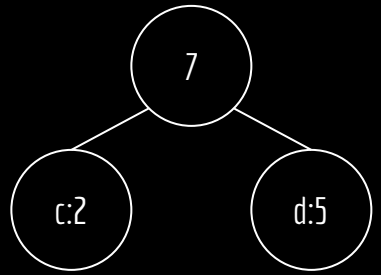
Teste de Mesa

Q

a	e	b	
6	6	7	7

huffman(C)

n	i
5	1

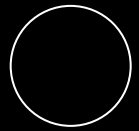


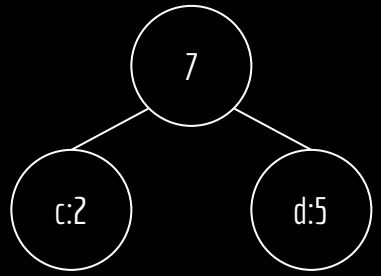

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

a	e	b	
6	6	7	7



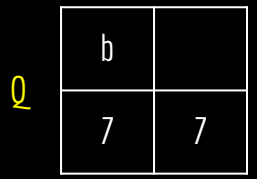


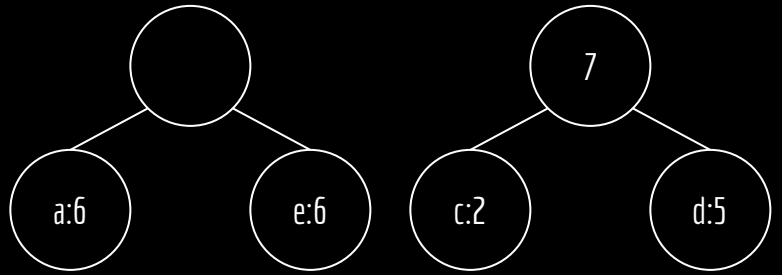
huffman(C)

n	i
5	1
	2

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa





huffman(C)

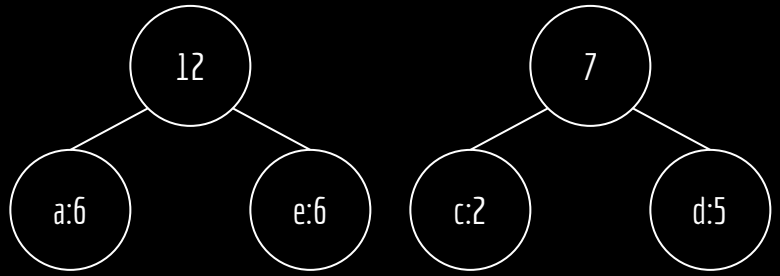
n	i
5	1
	2

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

b	
7	7



huffman(C)

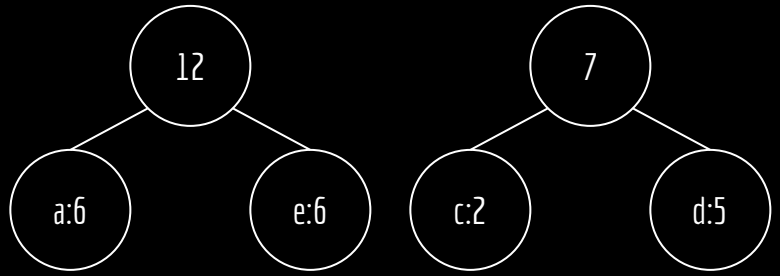
n	i
5	1
	2

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

b		
7	7	12



huffman(C)

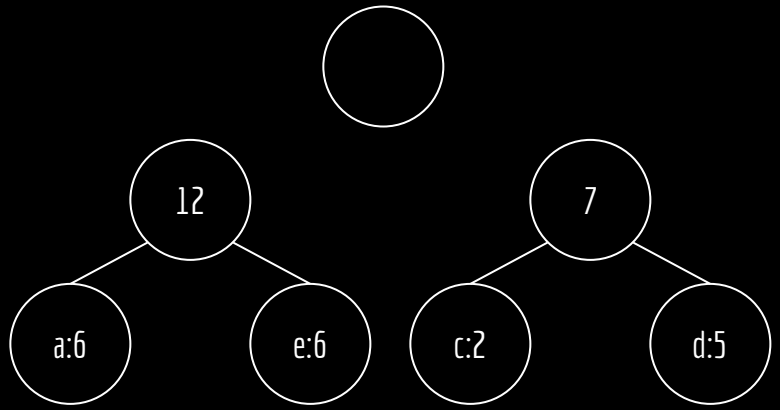
n	i
5	1
	2

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Teste de Mesa

Q

b		
7	7	12

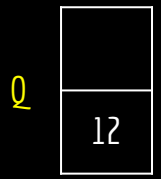


huffman(C)

n	i
5	1
	2
	3

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

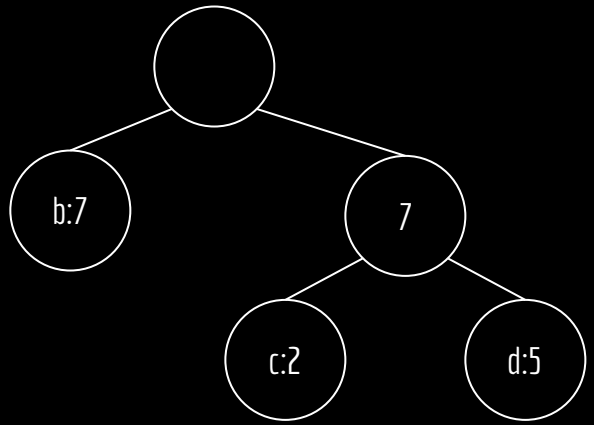
Teste de Mesa

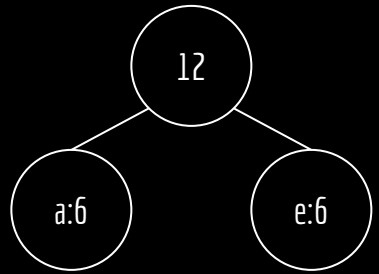


huffman(C)

n	i
5	1
	2
	3

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```





Teste de Mesa

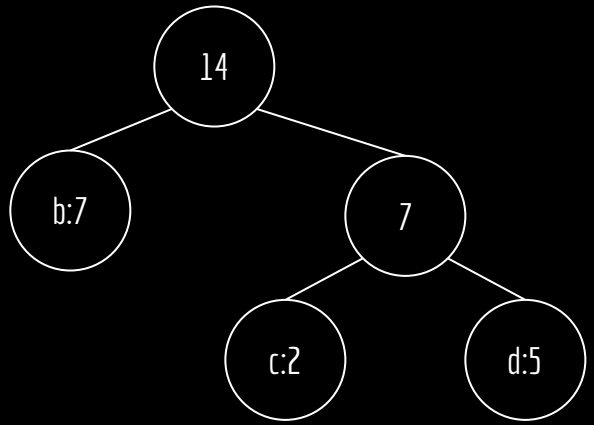
Q

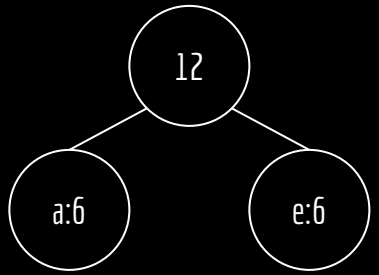
12	14

huffman(C)

n	i
5	1
	2
	3

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

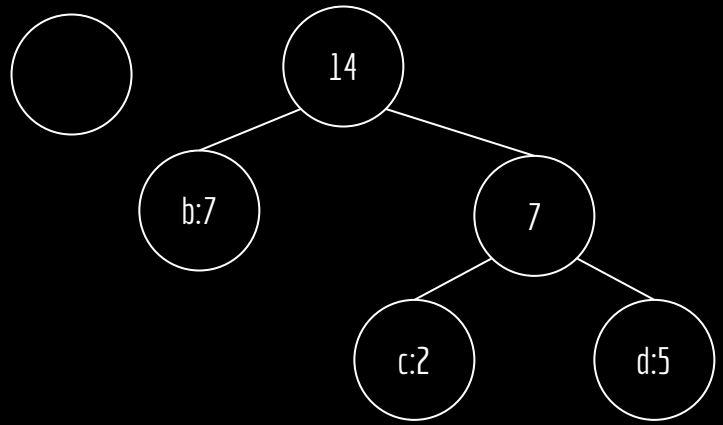




Teste de Mesa

Q

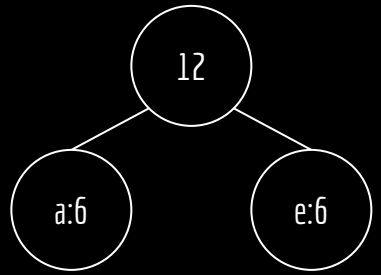
12	14



huffman(C)

n	i
5	1
	2
	3
	4

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

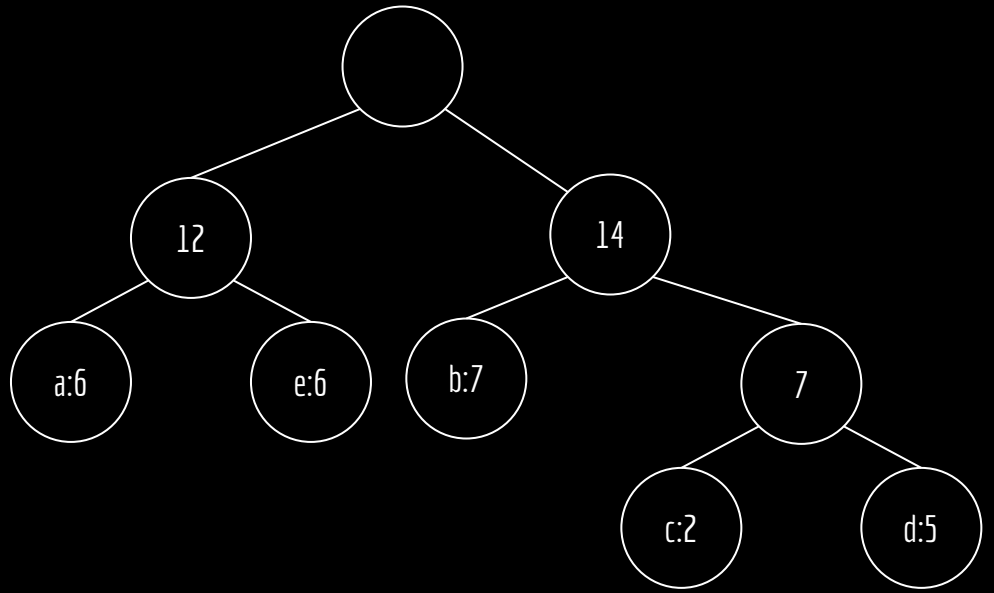


Teste de Mesa

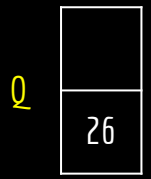
Q

huffman(C)
n i
5 1
2
3
4

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```



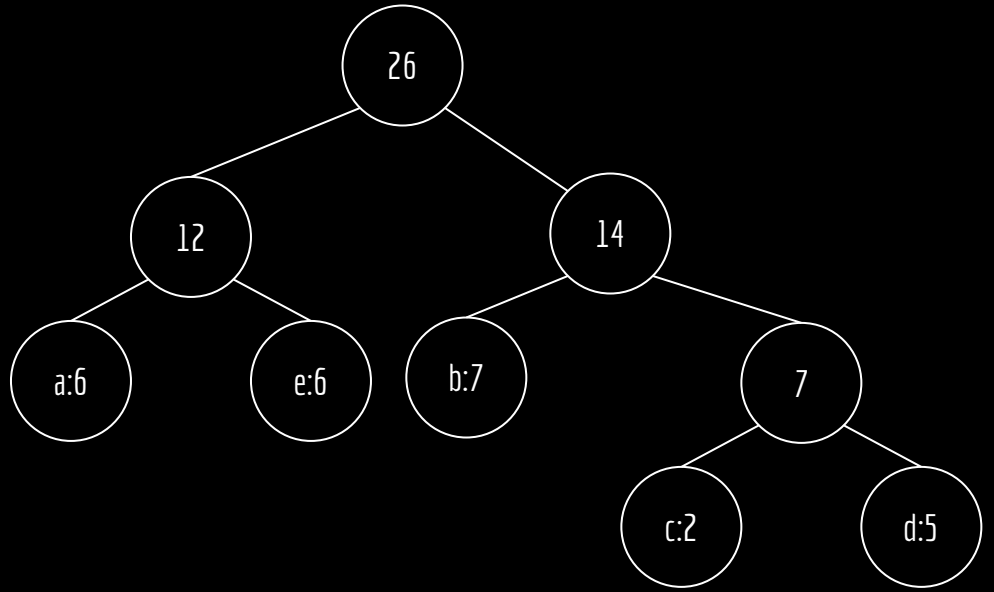
Teste de Mesa



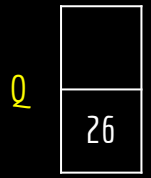
huffman(C)

n	i
5	1
	2
	3
	4

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```



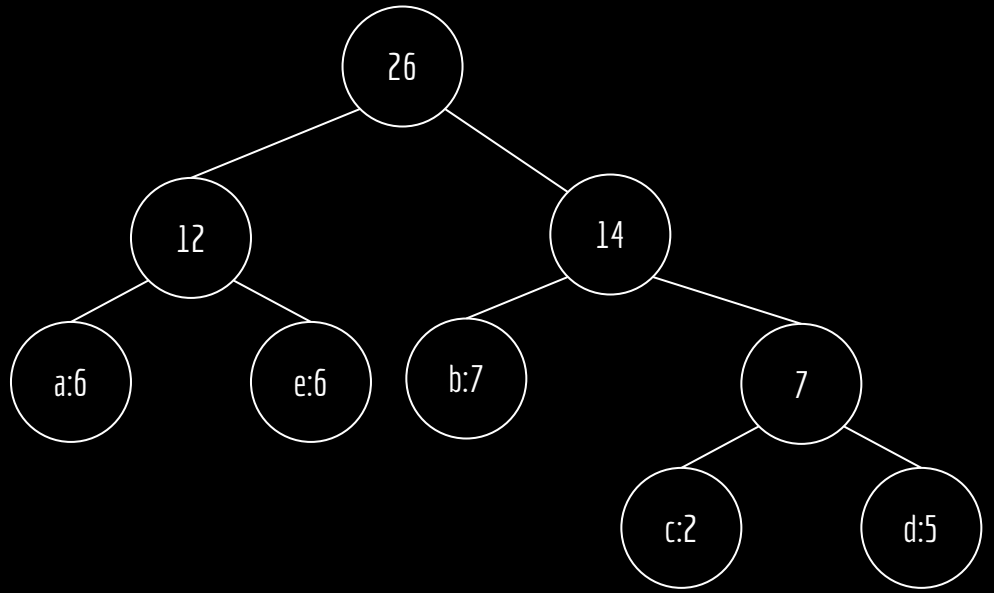
Teste de Mesa



huffman(C)

n	i
5	1
	2
	3
	4

```
função huffman(C)
n = |C|
Q = fila ordenada de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```



Características

O algoritmo é guloso.

O algoritmo sempre constrói uma árvore ótima e cheia.

Veja a prova em Cormen et al. (2022).

Custo

Para um texto com *n* caracteres, construir o histograma *C* custa *O(n)*.

```
função huffman(C)
n = |C|
Q = min-heap de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Custo

Se usar uma Min-Heap, construir a Heap tem um custo *O(n)*.

```
função huffman(C)
n = |C|
Q = min-heap de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Custo

```
função huffman(C)
n = |C|
Q = min-heap de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

Loop executado exatamente $n-1$ vezes.
As operações na Heap custam $O(log_2 n)$.

Custo

```
função huffman(C)
n = |C|
Q = min-heap de C
para i = 1 até n -1
     z = criar novo nodo
     z.fe = extrair mínimo de Q
     z.fd = extrair mínimo de Q
     z.freq = z.fe.freq + z.fd.freq
     inserir(Q,z)
retorne cabeça de Q
```

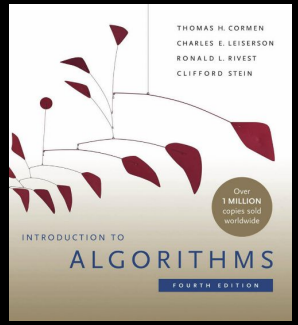
Logo, o algoritmo tem um custo $O(n \log_2 n)$.

Exercícios

1. Implemente o algoritmo do código de Huffman em C.
2. Use o algoritmo em 1 para que, dado um texto ASCII, salvá-lo em sua forma comprimida no disco.
3. Use o algoritmo em 1 para que, dado um arquivo texto comprimido, exibir seu conteúdo original.
4. Crie um algoritmo que, dada uma árvore de Código de Huffman, calcula o custo em bits para armazenar o arquivo.

Referências

T. Cormen, C. Leiserson, R. Rivest, C. Stein. Algoritmos: Teoria e Prática. 4a ed. 2022.



Estrutura de Dados e Algoritmos em C++. A. Drozdek. 4a ed. 2016.

Licença