Compactação de Dados

Letícia Rodrigues Bueno

UFABC

Compactação de Dados: Introdução

Compactação de Dados: Introdução

- **Objetivo:**

Compactação de Dados: Introdução

- **Objetivo:**
  1. minimizar espaço de memória utilizado;

Compactação de Dados: Introdução

- **Objetivo:**
  1. minimizar espaço de memória utilizado;
  2. minimizar custo da transmissão dos arquivos na rede ou tempo de leitura do arquivo no disco;

Compactação de Dados: Introdução

- **Objetivo:**
  1. minimizar espaço de memória utilizado;
  2. minimizar custo da transmissão dos arquivos na rede ou tempo de leitura do arquivo no disco;
  3. minimizar tempo de pesquisa no arquivo.

Compactação de Dados: Introdução

Compactação de Dados: Introdução

- **Possível solução:** substituir símbolos do texto por outros com menor número de *bits* ou *bytes*

Compactação de Dados: Introdução

- **Possível solução:** substituir símbolos do texto por outros com menor número de *bits* ou *bytes* (**codificação de mensagens**);

Compactação de Dados: Introdução

- **Possível solução:** substituir símbolos do texto por outros com menor número de *bits* ou *bytes* (**codificação de mensagens**);
  1. Dada uma cadeia de caracteres (mensagem);

Compactação de Dados: Introdução

- **Possível solução:** substituir símbolos do texto por outros com menor número de *bits* ou *bytes* (**codificação de mensagens**);
  1. Dada uma cadeia de caracteres (mensagem);
  2. **problema:** codificar mensagem através de atribuição de códigos a símbolos;

Compactação de Dados: Introdução

- **Possível solução:** substituir símbolos do texto por outros com menor número de *bits* ou *bytes* (**codificação de mensagens**);
  1. Dada uma cadeia de caracteres (mensagem);
  2. **problema:** codificar mensagem através de atribuição de códigos a símbolos;
  3. tabela de códigos armazenada para decodificação;

Algoritmo de Frequência de Caracteres

Algoritmo de Frequência de Caracteres

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC

Algoritmo de Frequência de Caracteres

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C

Algoritmo de Frequência de Caracteres

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

Algoritmo de Frequência de Caracteres

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- texto não pode ter caracteres numéricos;

Algoritmo de Frequência de Caracteres

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- texto não pode ter caracteres numéricos;
- pode-se utilizar uma representação para diferenciar símbolos e frequências;

Algoritmo de Frequência de Caracteres

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- texto não pode ter caracteres numéricos;
- pode-se utilizar uma representação para diferenciar símbolos e frequências;

AAA33333BA6666888DDDDDDD99999999999AABBB

Algoritmo de Frequência de Caracteres

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- texto não pode ter caracteres numéricos;
- pode-se utilizar uma representação para diferenciar símbolos e frequências;

AAA33333BA6666888DDDDDDD99999999999AABBB
3A5@3BA4@63@87D11@92A3B

Algoritmo de Huffman

- proposto em 1952;

Algoritmo de Huffman

- proposto em 1952;
- **ideia básica:** atribuir códigos mais curtos a símbolos com frequências altas;

Algoritmo de Huffman

- proposto em 1952;
- **ideia básica:** atribuir códigos mais curtos a símbolos com frequências altas;
- **implementação tradicional:** considera caractere como símbolo (comprime texto em $\approx 25\%$);

Algoritmo de Huffman

- proposto em 1952;
- **ideia básica:** atribuir códigos mais curtos a símbolos com frequências altas;
- **implementação tradicional:** considera caractere como símbolo (comprime texto em $\approx$ 25%);
- considerar palavras como símbolos comprime texto em $\approx$ 60%;

Algoritmo de Huffman

- Conjunto de símbolos: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;

Algoritmo de Huffman

- Conjunto de símbolos: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i é frequência de s_i no texto, para $1 \leq i \leq n$;

Algoritmo de Huffman

- Conjunto de símbolos: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i é frequência de s_i no texto, para $1 \leq i \leq n$;
- Queremos atribuir um código a cada símbolo para compactar o texto todo;

Algoritmo de Huffman

- Conjunto de símbolos: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i é frequência de s_i no texto, para $1 \leq i \leq n$;
- Queremos atribuir um código a cada símbolo para compactar o texto todo;
- Nenhum código deve ser prefixo de outro: árvore binária de prefixo;

Algoritmo de Huffman

- Conjunto de símbolos: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i é frequência de s_i no texto, para $1 \leq i \leq n$;
- Queremos atribuir um código a cada símbolo para compactar o texto todo;
- Nenhum código deve ser prefixo de outro: árvore binária de prefixo;
- Cada símbolo s_i é associado a uma folha da árvore;

Algoritmo de Huffman

- Conjunto de símbolos: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i é frequência de s_i no texto, para $1 \leq i \leq n$;
- Queremos atribuir um código a cada símbolo para compactar o texto todo;
- Nenhum código deve ser prefixo de outro: árvore binária de prefixo;
- Cada símbolo s_i é associado a uma folha da árvore;
- Códigos dos símbolos são sequências binárias;

Algoritmo de Huffman

- Conjunto de símbolos: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i é frequência de s_i no texto, para $1 \leq i \leq n$;
- Queremos atribuir um código a cada símbolo para compactar o texto todo;
- Nenhum código deve ser prefixo de outro: árvore binária de prefixo;
- Cada símbolo s_i é associado a uma folha da árvore;
- Códigos dos símbolos são sequências binárias;
- **Vantagem de utilizar código de prefixo:** facilidade para codificação e decodificação;

Algoritmo de Huffman: (de)codificação

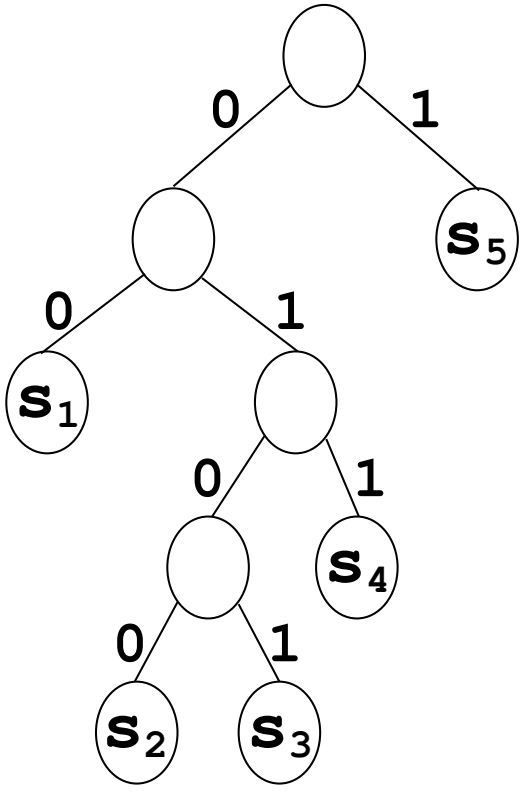
Algoritmo de Huffman: (de)codificação

$s_4\, s_3\, s_3\, s_1\, s_3\, s_1\, s_4\, s_5\, s_1\, s_3\, s_3\, s_3\, s_3\, s_3\, s_2\, s_3\, s_5\, s_2\, s_2\, s_2\, s_4$

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

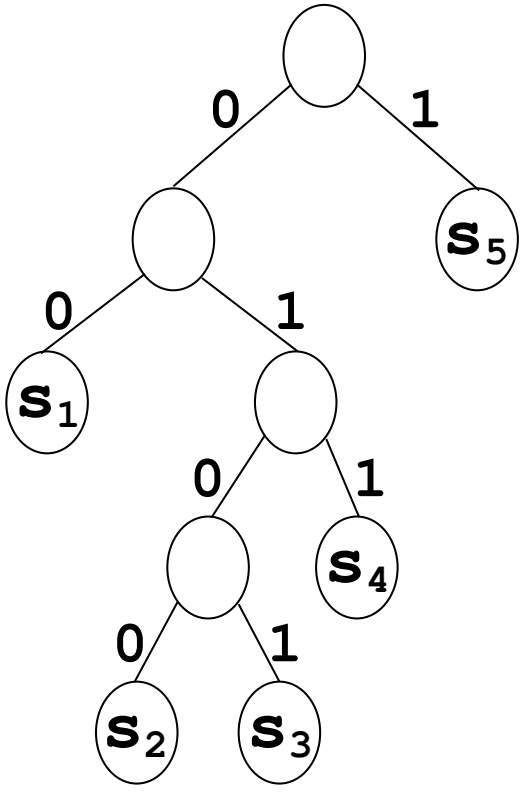
s_1:00
s_2:0100
s_3:0101
s_4:011
s_5:1



Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

s_1:00
s_2:0100
s_3:0101
s_4:011
s_5:1

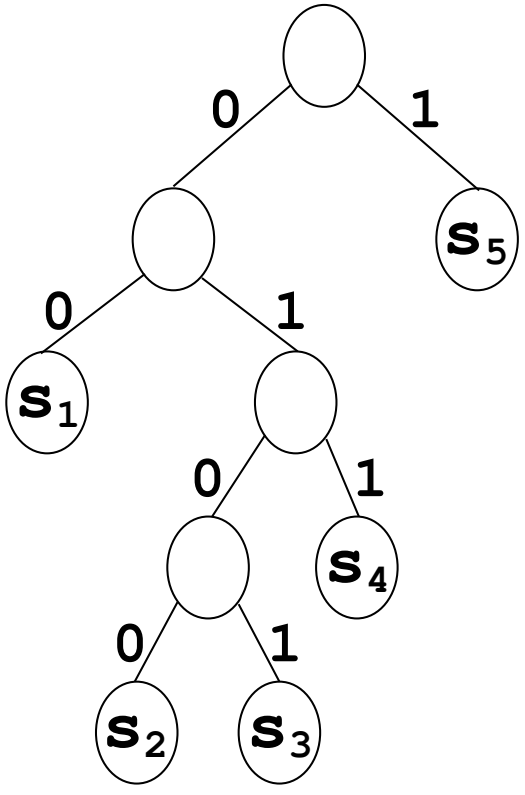


011

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

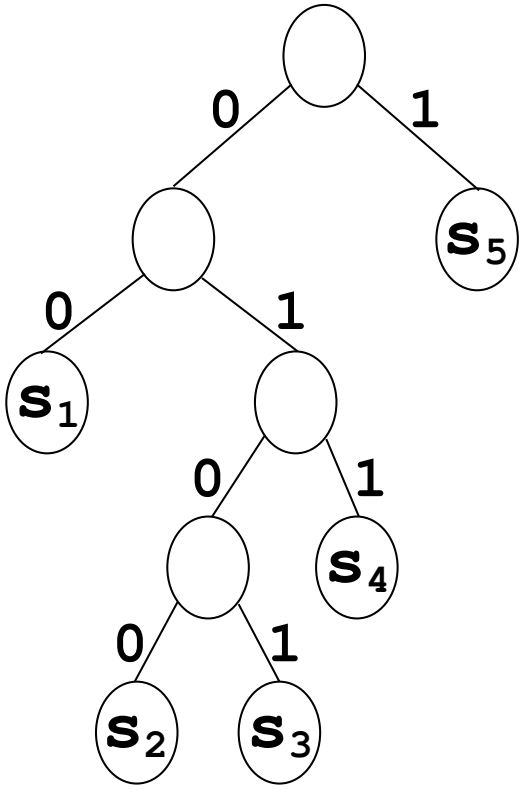


011 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

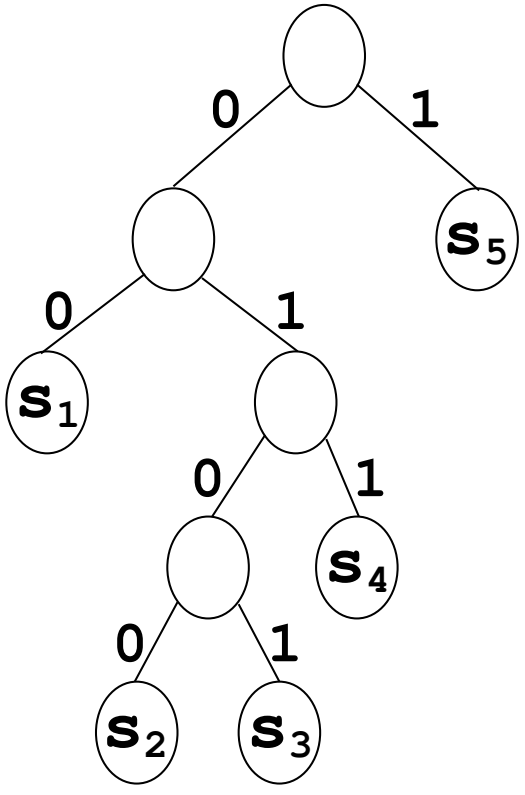


011 0101 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

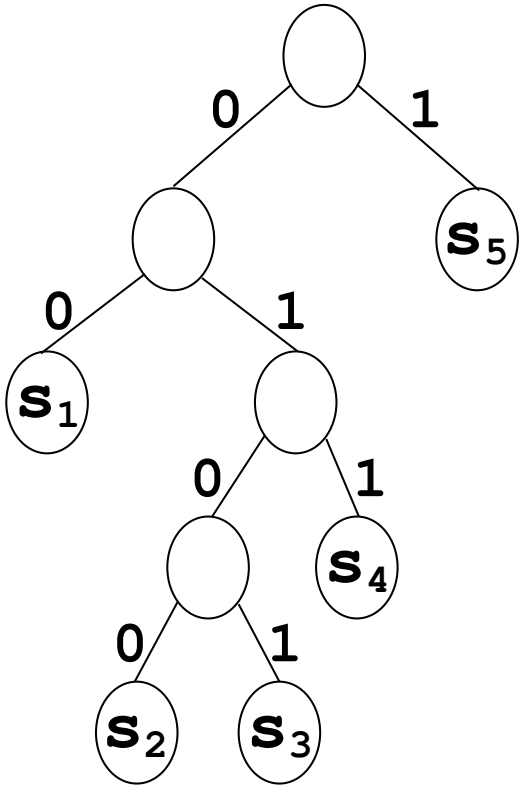


011 0101 0101 00

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

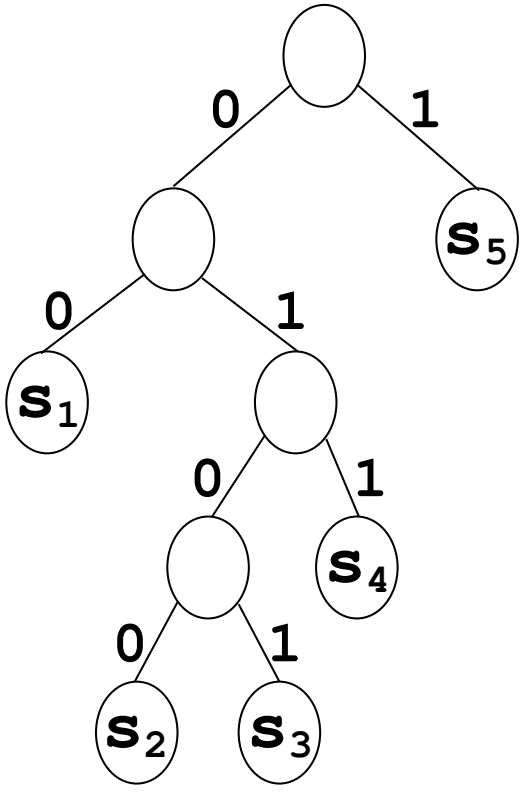


011 0101 0101 00 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

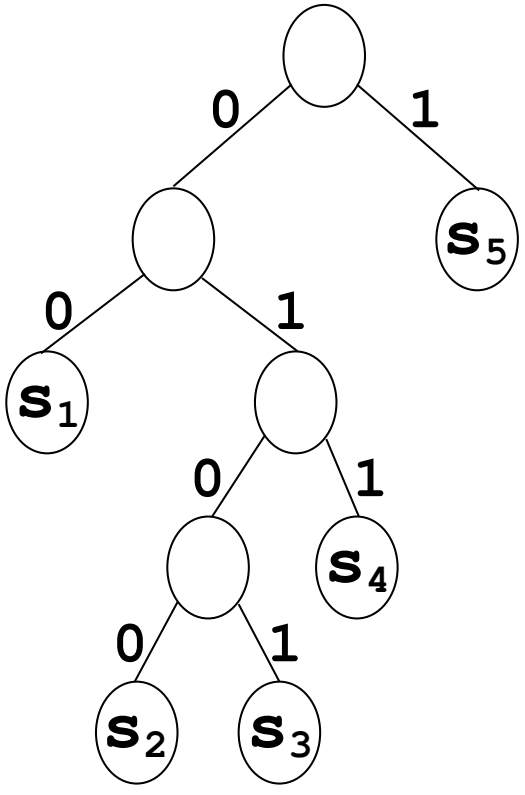


011 0101 0101 00 0101 00

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

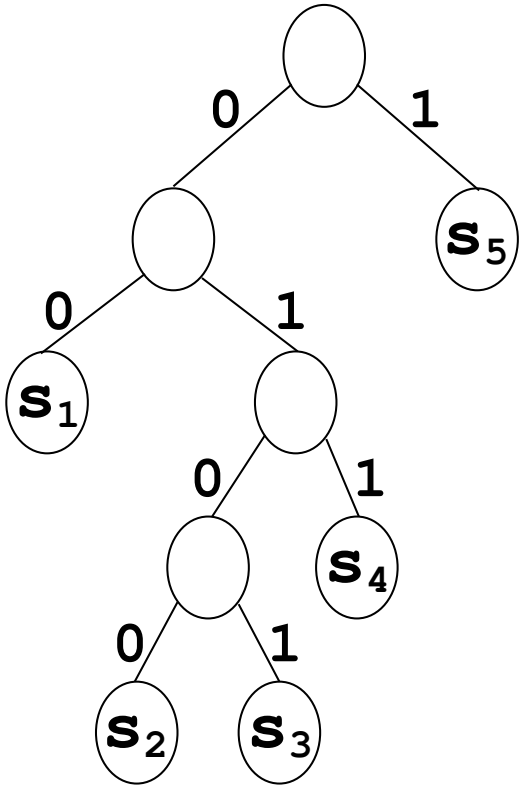


011 0101 0101 00 0101 00 011

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

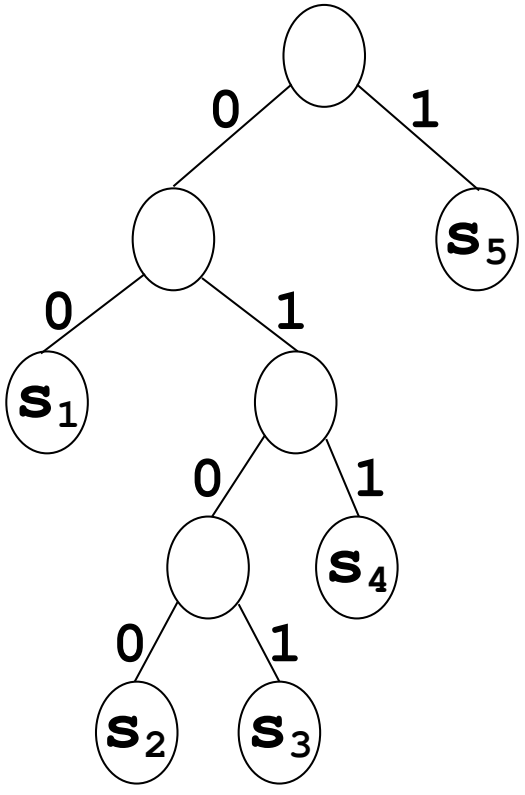


011 0101 0101 00 0101 00 011 1

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

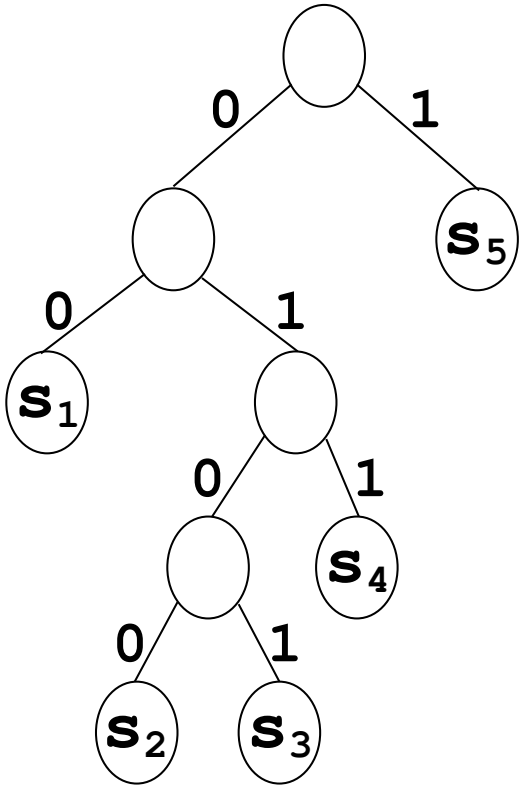


011 0101 0101 00 0101 00 011 1 00

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

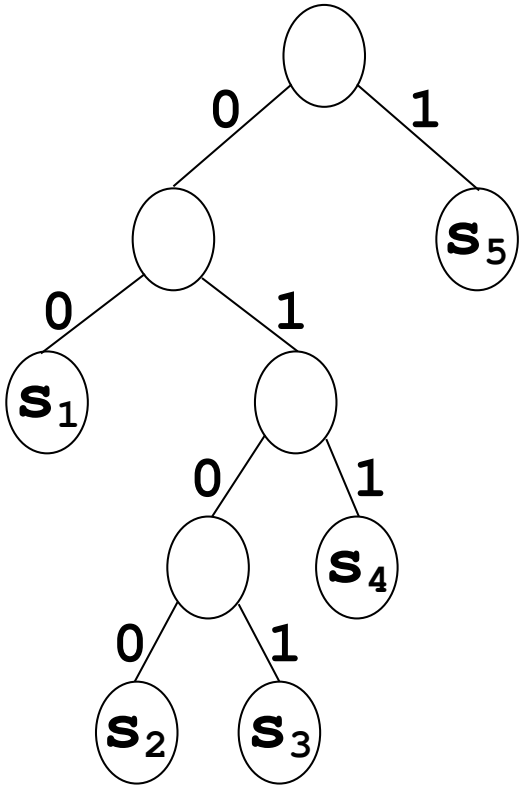


011 0101 0101 00 0101 00 011 1 00 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

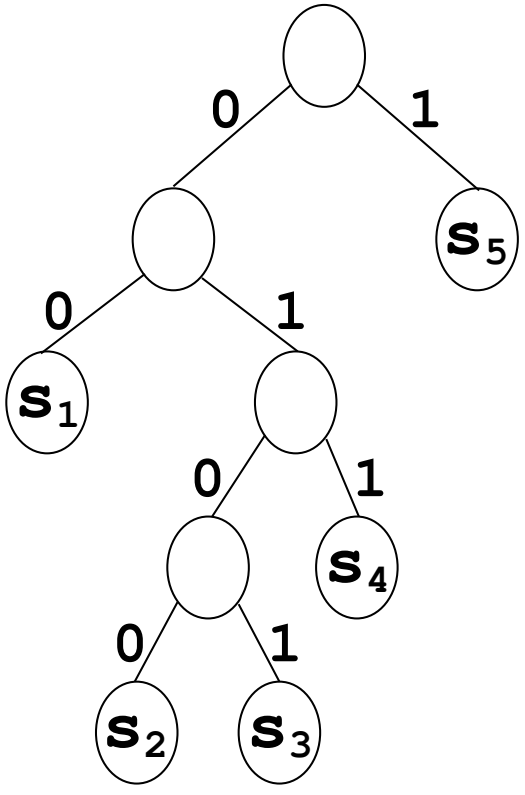


011 0101 0101 00 0101 00 011 1 00 0101 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

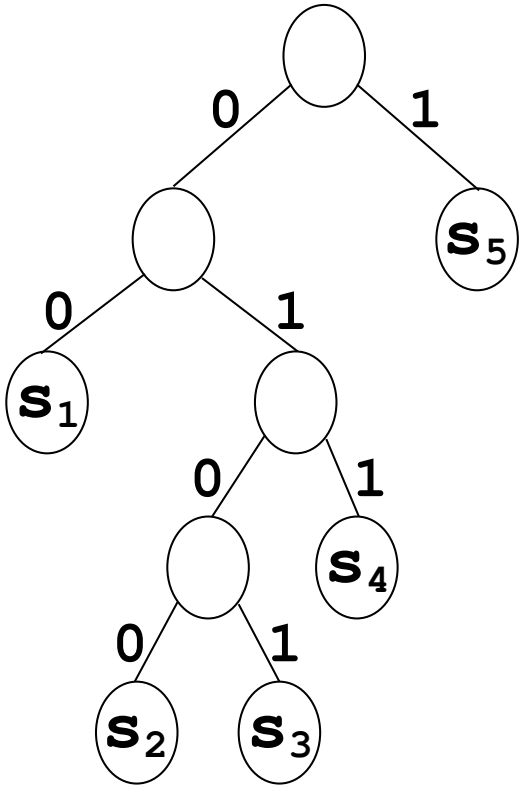


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

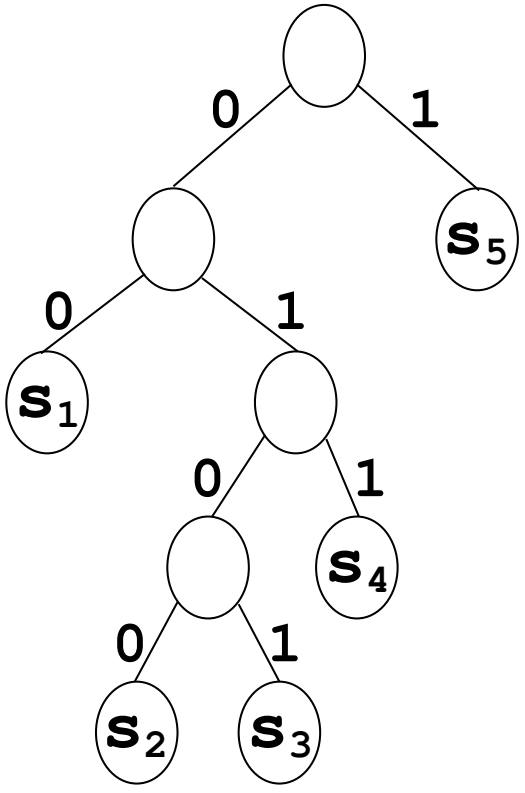


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

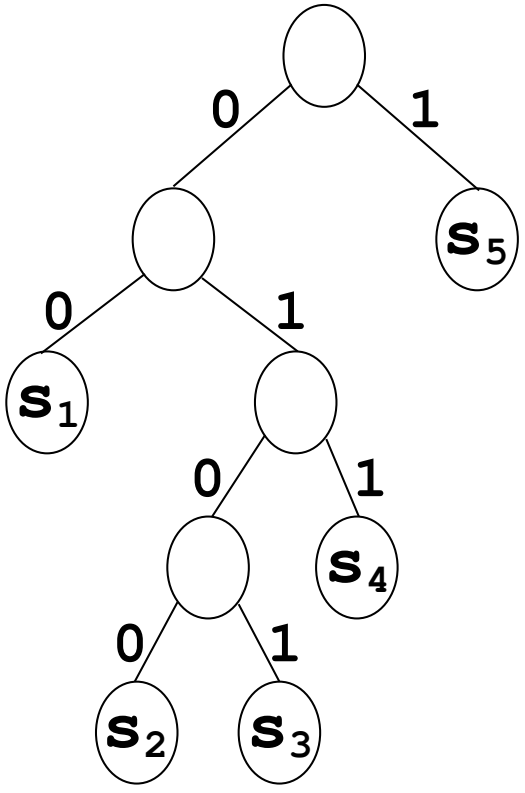


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

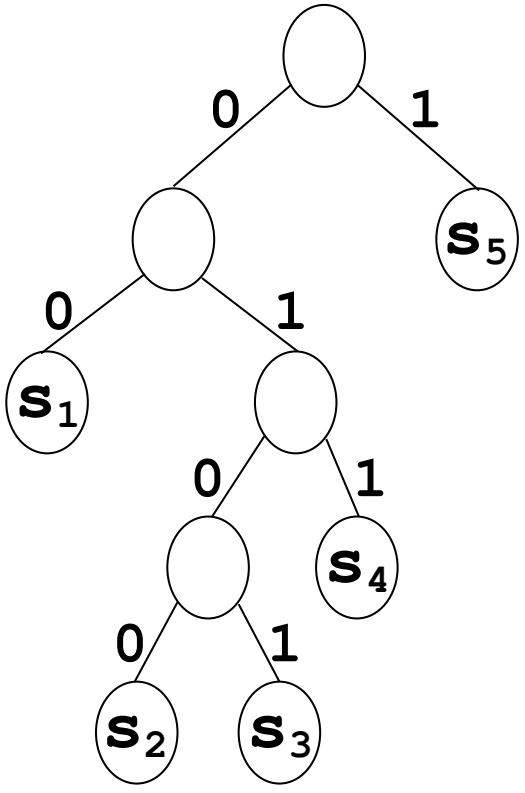


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

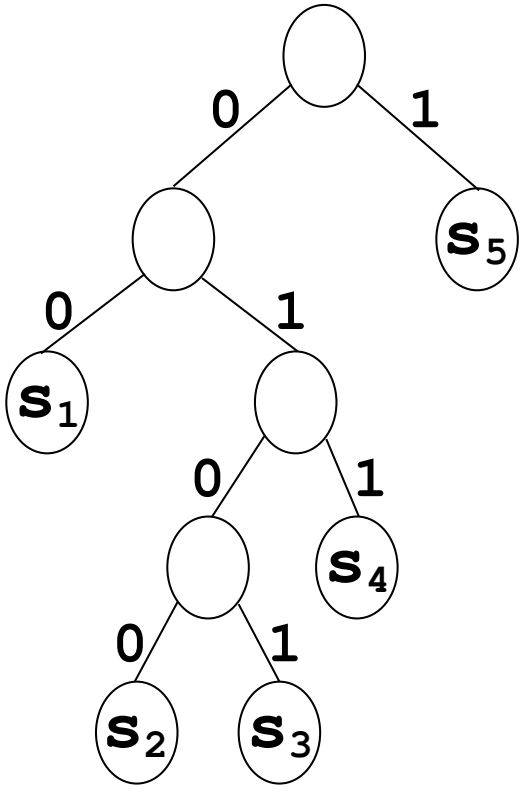


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

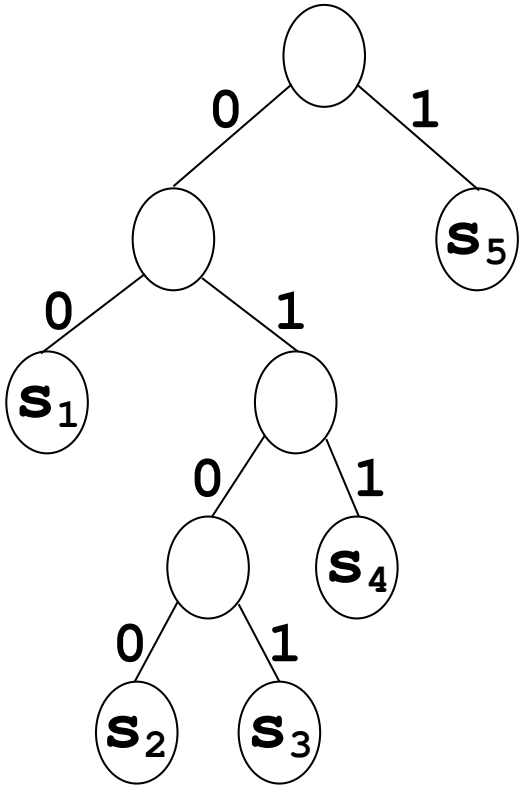


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

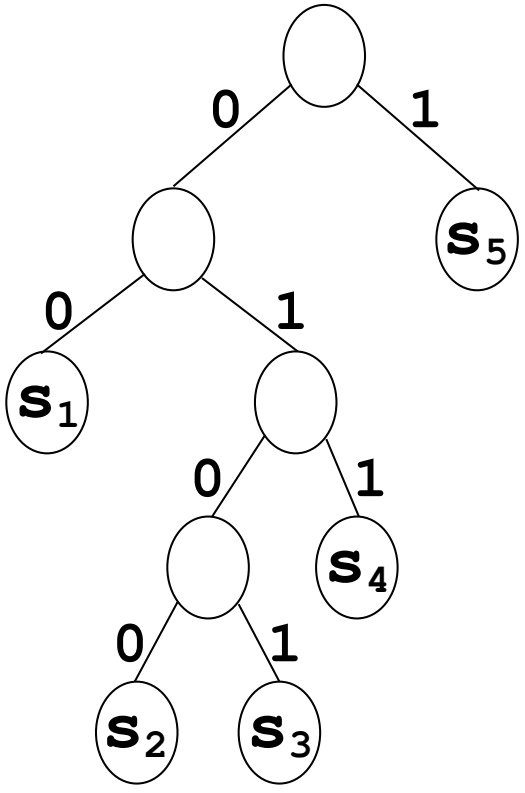


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1 0100

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

s_1:00
s_2:0100
s_3:0101
s_4:011
s_5:1

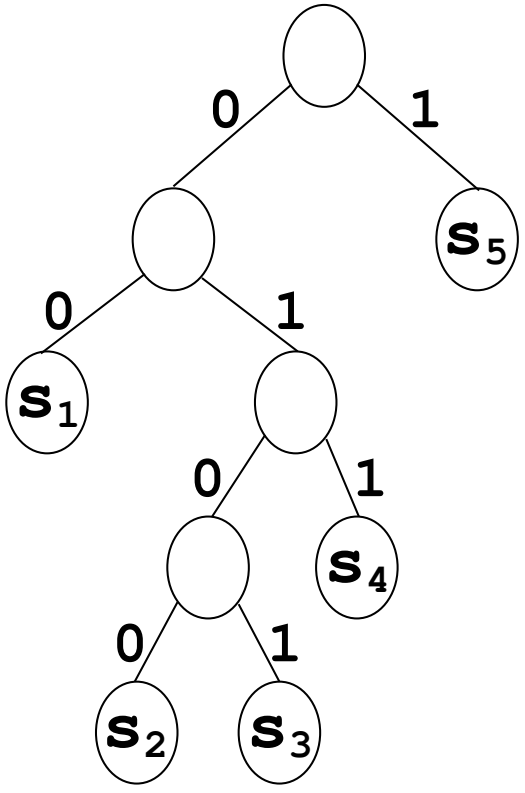


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1 0100 0100

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$

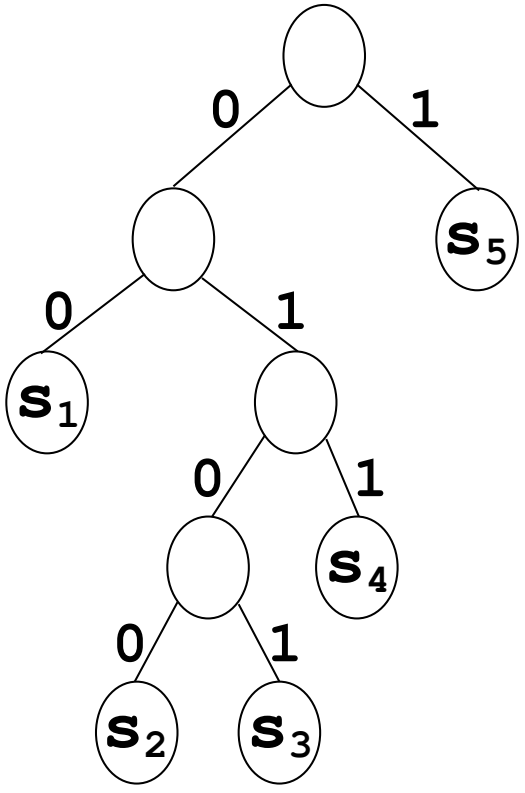


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1 0100 0100 0100

Algoritmo de Huffman: (de)codificação

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$

$s_1 : 00$
$s_2 : 0100$
$s_3 : 0101$
$s_4 : 011$
$s_5 : 1$



011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1 0100 0100 0100 011

Algoritmo de Huffman

- Se árvore de prefixo T é conhecida, (de)codificação é feita em $O(m)$ onde m é o tamanho da sequência binária codificada;

Algoritmo de Huffman

- Se árvore de prefixo T é conhecida, (de)codificação é feita em $O(m)$ onde m é o tamanho da sequência binária codificada;
- Para 01101010101000101000111000101010101010101010101000101101010001000100011, custo $c(T) = 69$;

Algoritmo de Huffman

- Se árvore de prefixo T é conhecida, (de)codificação é feita em $O(m)$ onde m é o tamanho da sequência binária codificada;
- Para 01101010101000101000111000101010101010101010101000101101010001000100011, custo $c(T) = 69$;
- É necessário minimizar $c(T)$;

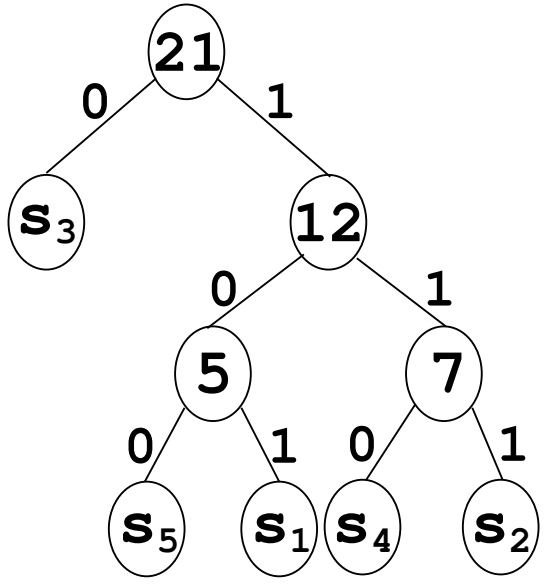
Algoritmo de Huffman

- Se árvore de prefixo T é conhecida, (de)codificação é feita em $O(m)$ onde m é o tamanho da sequência binária codificada;
- Para 01101010101000101000111000101010101010101000101011010001000100011, custo $c(T) = 69$;
- É necessário minimizar $c(T)$;
- **Árvore mínima** ou **árvore de Huffman**: árvore de prefixo T com $c(T)$ mínimo;

Algoritmo de Huffman

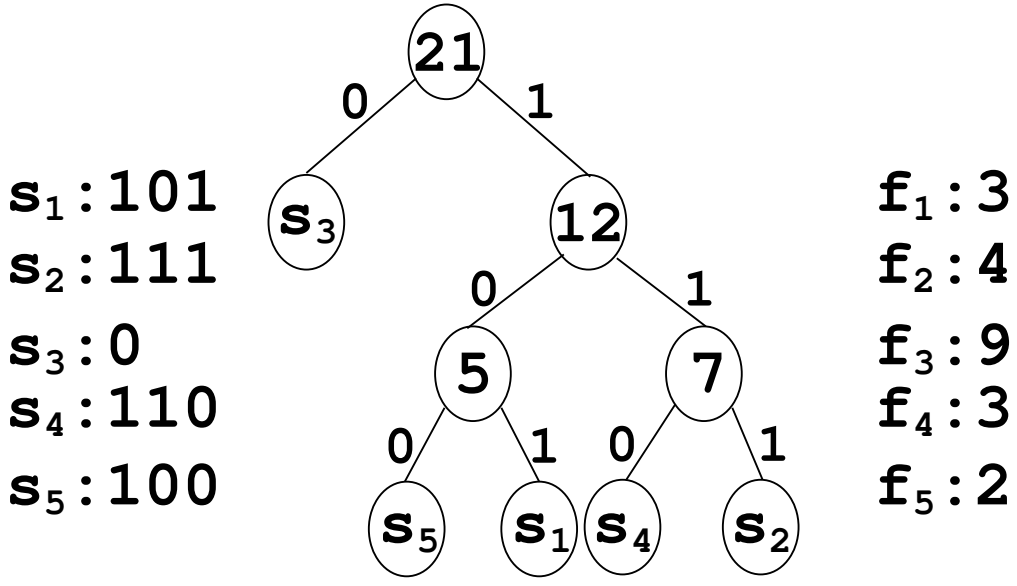
Algoritmo de Huffman

s_1:101
s_2:111
s_3:0
s_4:110
s_5:100



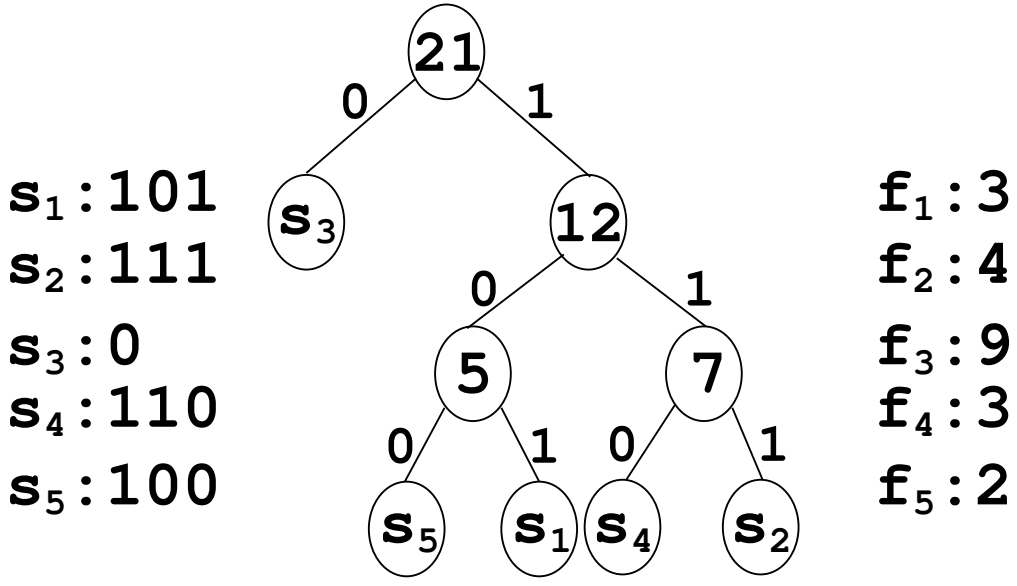
f_1:3
f_2:4
f_3:9
f_4:3
f_5:2

Algoritmo de Huffman



$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$ produz

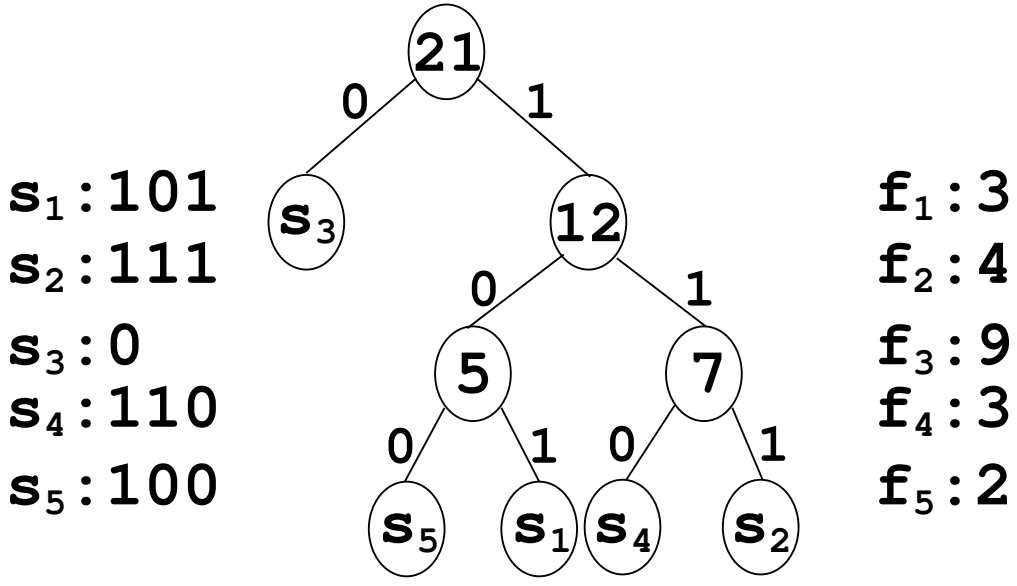
Algoritmo de Huffman



$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$ produz

1100010101011101001010000011101001111111111110

Algoritmo de Huffman



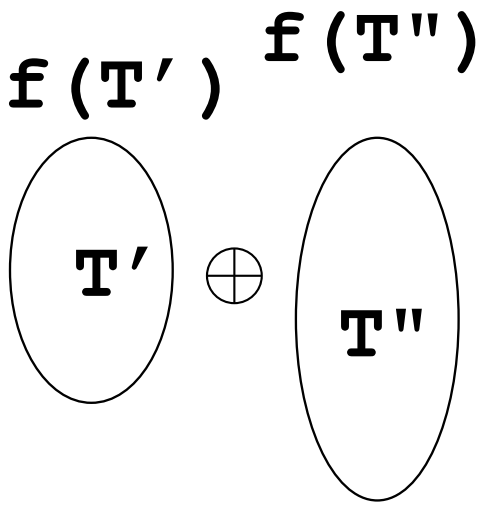
$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$ produz

110001010101111010010100000111010011111111110

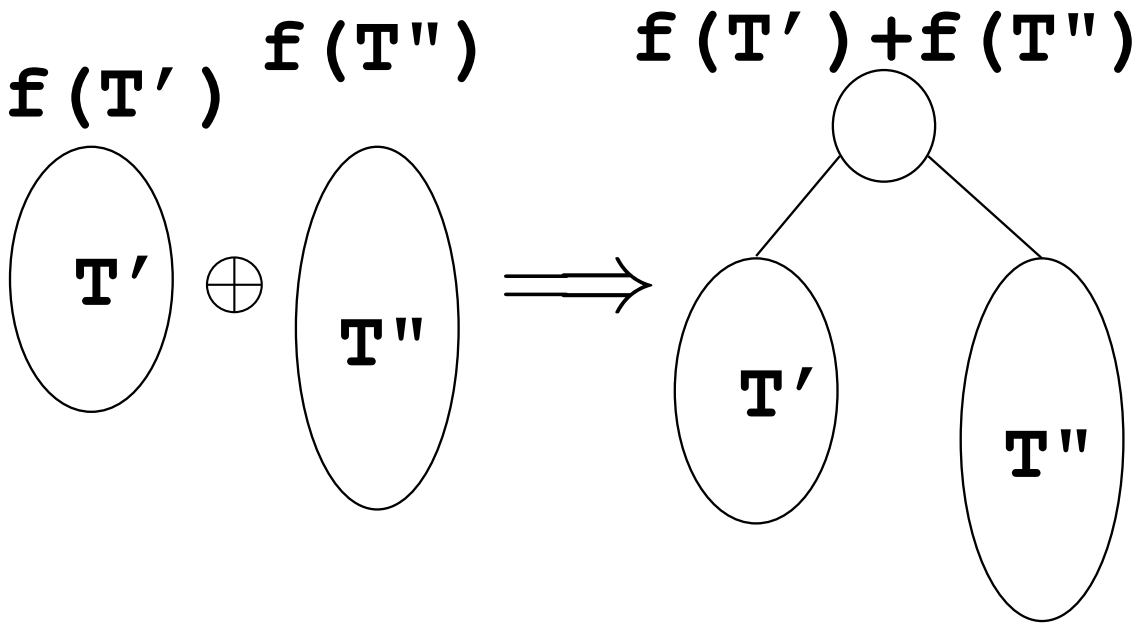
que tem $c(T) = 45$ e que é mínima.

Ideia do Algoritmo de Huffman

Ideia do Algoritmo de Huffman

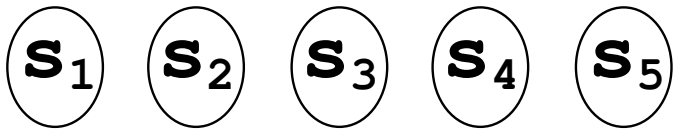


Ideia do Algoritmo de Huffman

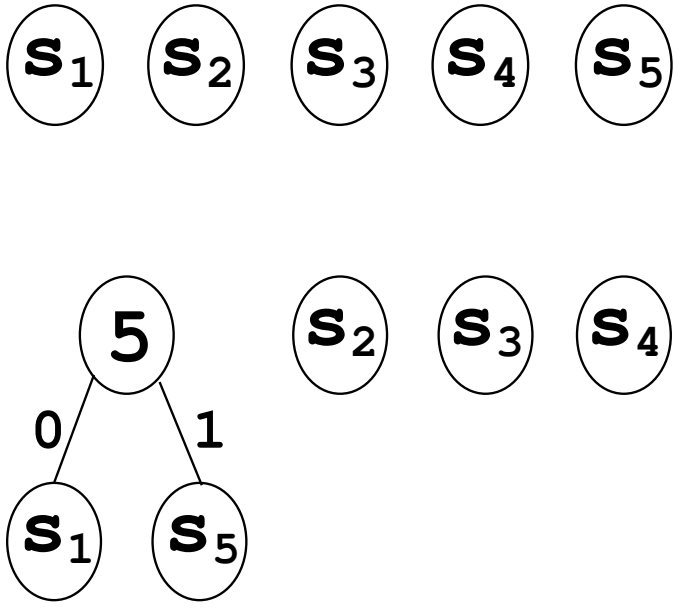


Construção da Árvore de Huffman

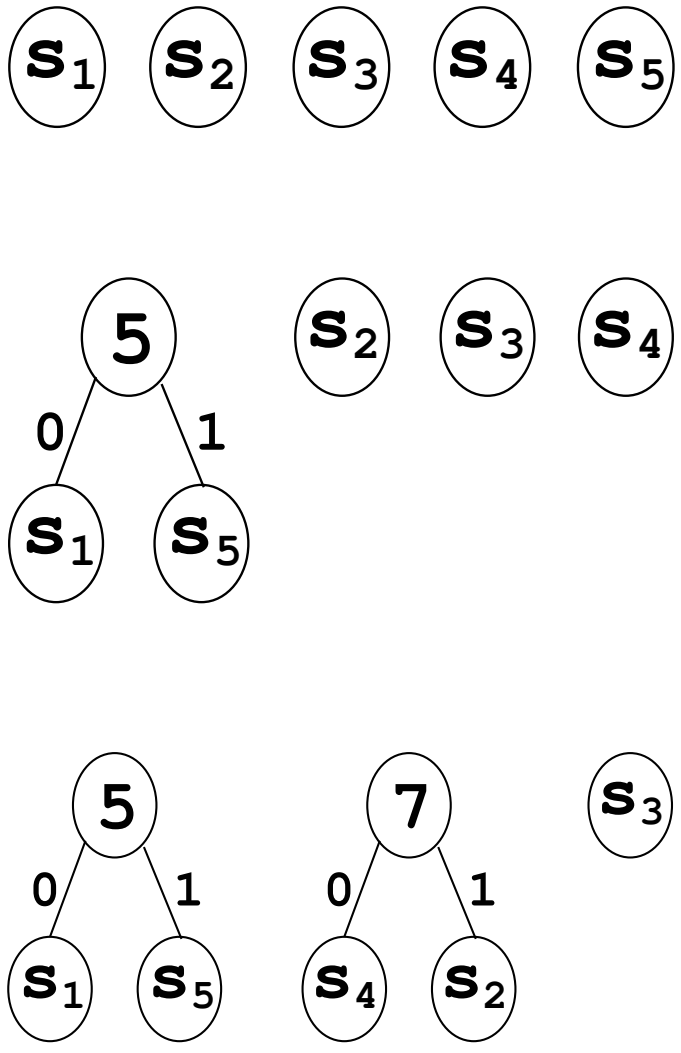
Construção da Árvore de Huffman



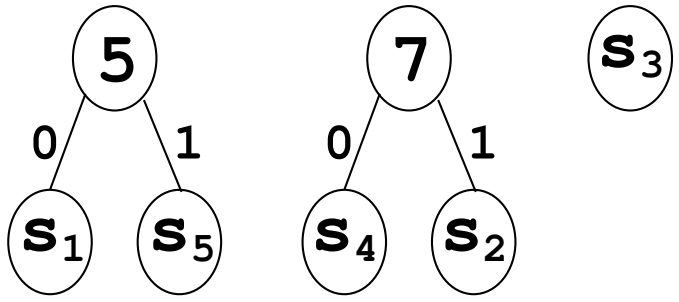
Construção da Árvore de Huffman



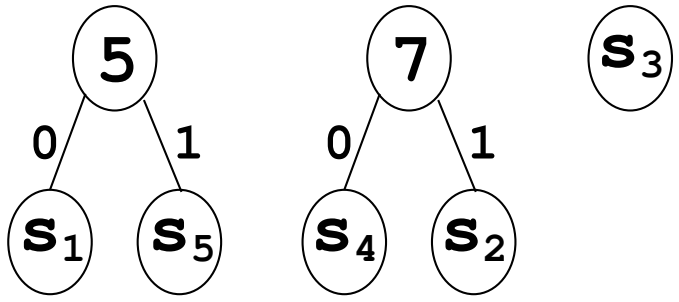
Construção da Árvore de Huffman

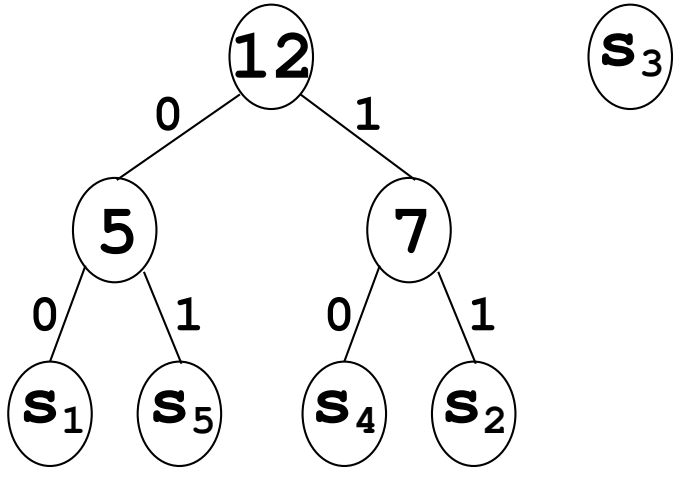


Construção da Árvore de Huffman

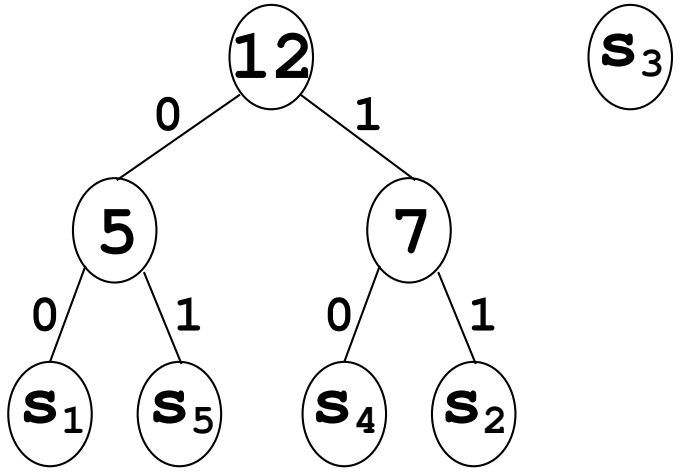


Construção da Árvore de Huffman

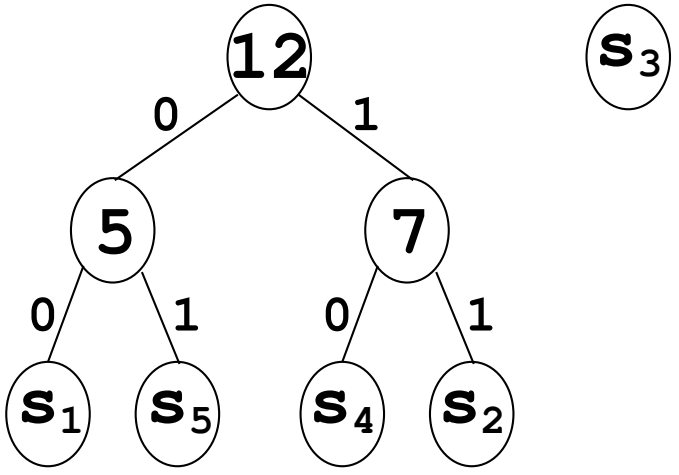


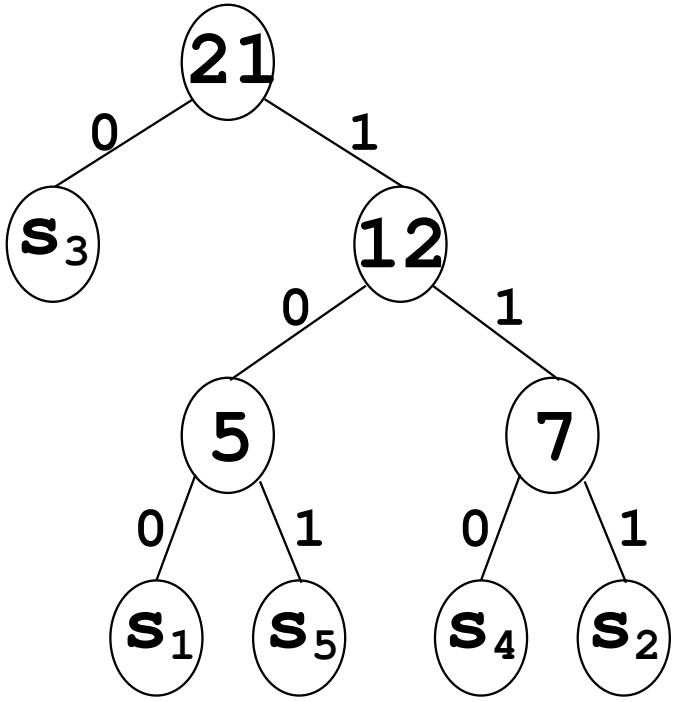


Construção da Árvore de Huffman



Construção da Árvore de Huffman





Algoritmo de Huffman: inserção em lista de prioridades

Algoritmo de Huffman: inserção em lista de prioridades

```
1  inserir(T, f, F):
2      F[n + 1] ← T
3      n++
4      subir(n)
```

Algoritmo de Huffman: inserção em lista de prioridades

```
1  inserir(T, f, F):
2      F[n + 1] ← T
3      n++
4      subir(n)
```

```
1  subir(i):
2      j = ⌊i/2⌋
3      se j ≥ 1 então
4          se T[i] < T[j] então
5              T[i] ⇔ T[j]
6              subir(j)
```

Algoritmo de Huffman: inserção em lista de prioridades

```
1  inserir(T, f, F):
2      F[n + 1] ← T
3      n++
4      subir(n)
```

```
1  subir(i):
2      j = ⌊i/2⌋
3      se j ≥ 1 então
4          se T[i] < T[j] então
5              T[i] ⇔ T[j]
6              subir(j)
```

- **inserir(T, f, F) e subir(i)**: métodos de lista de prioridades implementada por *heap*;

Algoritmo de Huffman: remoção de lista de prioridades

Algoritmo de Huffman: remoção de lista de prioridades

```
1  minimo(T, f, F):
2      se n ≠ 0 então
3          remove T[1]
4          T[1] ← T[n]
5          n- -
6          descer(1, n)
```

Algoritmo de Huffman: remoção de lista de prioridades

```
1  minimo(T, f, F):
2      se n ≠ 0 então
3          remove T[1]
4          T[1] ← T[n]
5          n- -
6          descer(1, n)
```

```
1  descer(i, n):
2      j = 2 * i
3      se j ≤ n então
4          se j < n então
5              se T[j + 1] < T[j] então
6                  j = j + 1
7          se T[i] > T[j] então
8              T[i] ⇔ T[j]
9              descer(j, n)
```

Algoritmo de Huffman: remoção de lista de prioridades

```
1  minimo(T, f, F):
2      se n ≠ 0 então
3          remove T[1]
4          T[1] ← T[n]
5          n- -
6          descer(1, n)
```

```
1  descer(i, n):
2      j = 2 * i
3      se j ≤ n então
4          se j < n então
5              se T[j + 1] < T[j] então
6                  j = j + 1
7          se T[i] > T[j] então
8              T[i] ⇔ T[j]
9              descer(j, n)
```

- **minimo(**T, f, F**) e descer(**$1, n$**)**: métodos de lista de prioridades implementada por *heap*;

Algoritmo de Huffman

```
1  huffman():
2      para i ← 1 até n − 1 faça
3          minimo(T′, f, F);
4          minimo(T″, f, F);
5          T ← T′ ⊕ T″;
6          f(T) ← f(T′) ⊕ f(T″);
7          inserir(T, f, F);
```

Algoritmo de Huffman

```
1  huffman():
2      para i ← 1 até n − 1 faça
3          minimo(T′, f, F);
4          minimo(T″, f, F);
5          T ← T′ ⊕ T″;
6          f(T) ← f(T′) ⊕ f(T″);
7          inserir(T, f, F);
```

- Operação de minimização ou inclusão na lista de prioridades: $O(\log n)$;

Algoritmo de Huffman

```
1  huffman():
2      para i ← 1 até n − 1 faça
3          minimo(T′, f, F);
4          minimo(T″, f, F);
5          T ← T′ ⊕ T″;
6          f(T) ← f(T′) ⊕ f(T″);
7          inserir(T, f, F);
```

- Operação de minimização ou inclusão na lista de prioridades: $O(\log n)$;
- operação $\oplus$ requer número constante de passos;

Algoritmo de Huffman

```
1  huffman():
2      para i ← 1 até n − 1 faça
3          minimo(T′, f, F);
4          minimo(T″, f, F);
5          T ← T′ ⊕ T″;
6          f(T) ← f(T′) ⊕ f(T″);
7          inserir(T, f, F);
```

- Operação de minimização ou inclusão na lista de prioridades: $O(\log n)$;
- operação $\oplus$ requer número constante de passos;
- Total de $n - 1$ iterações;

Algoritmo de Huffman

```
1  huffman():
2      para i ← 1 até n − 1 faça
3          minimo(T′, f, F);
4          minimo(T″, f, F);
5          T ← T′ ⊕ T″;
6          f(T) ← f(T′) ⊕ f(T″);
7          inserir(T, f, F);
```

- Operação de minimização ou inclusão na lista de prioridades: $O(\log n)$;
- operação $\oplus$ requer número constante de passos;
- Total de $n - 1$ iterações;
- **Complexidade:** $O(n \log n)$;

Exercícios

1. Desenhar a árvore de Huffman para o seguinte conjunto de chaves e frequências, respectivamente na primeira e segunda linhas da tabela:

s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
1	6	2	1	1	9	2	3

2. A árvore fornecida pelo algoritmo de Huffman é única, ou seja, o algoritmo sempre fornecerá a mesma árvore?

Bibliografia

SZWARCFITER, J. L. e MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994.

ZIVIANI, N. Projeto de Algoritmos: com implementações em Java e C++, 1a edição, Cengage Learning, 2009.

Perguntas?