Data Compression

Letícia Rodrigues Bueno

Federal University of ABC (UFABC)

Data Compression: Introduction

Data Compression: Introduction

- **Objectives:**

Data Compression: Introduction

- **Objectives:**
  1. minimize memory space;

Data Compression: Introduction

- **Objectives:**
  1. minimize memory space;
  2. minimize the cost for sending files through the network or the time for reading files from disks;

Data Compression: Introduction

- **Objectives:**
  1. minimize memory space;
  2. minimize the cost for sending files through the network or the time for reading files from disks;
  3. minimize time of searching inside files.

Data Compression: Introduction

Data Compression: Introduction

- **Possible solution:** replace symbols throughout the text by others with smaller number of bits or bytes

- **Possible solution:** replace symbols throughout the text by others with smaller number of bits or bytes (**message encoding**);

Data Compression: Introduction

- **Possible solution:** replace symbols throughout the text by others with smaller number of bits or bytes (**message encoding**);
    1. Given a string of characters (message);

Data Compression: Introduction

- **Possible solution:** replace symbols throughout the text by others with smaller number of bits or bytes (**message encoding**);
    1. Given a string of characters (message);
    2. **Problem:** encode message by assigning codes to symbols;

Data Compression: Introduction

- **Possible solution:** replace symbols throughout the text by others with smaller number of bits or bytes (**message encoding**);
    1. Given a string of characters (message);
    2. **Problem:** encode message by assigning codes to symbols;
    3. table of codes is stored for decoding;

Run-length encoding (RLE)

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC

3B1E4A2F5H1C1B2M1A3L1C2D7B2C

3BE4A2F5HCB2MA3LC2D7B2C

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- text cannot have numerical characters;

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- text cannot have numerical characters;
- we can use a representation to distinguish symbols to frequencies;

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- text cannot have numerical characters;
- we can use a representation to distinguish symbols to frequencies;

Example:

AAA33333BA6666888DDDDDDD9999999999AABBB

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- text cannot have numerical characters;
- we can use a representation to distinguish symbols to frequencies;

Example:

AAA33333BA6666888DDDDDDD99999999999AABBB
becomes

Run-length encoding (RLE)

BBBEAAAAFFHHHHHCBMMALLLCDDBBBBBBBCC
3B1E4A2F5H1C1B2M1A3L1C2D7B2C
3BE4A2F5HCB2MA3LC2D7B2C

- text cannot have numerical characters;
- we can use a representation to distinguish symbols to frequencies;

Example:

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

Huffman's Algorithm

- proposed in 1952;

Huffman's Algorithm

- proposed in 1952;
- **main idea:** assign shorter codes (fewer bits) to symbols with high frequencies;

Huffman's Algorithm

- proposed in 1952;
- **main idea:** assign shorter codes (fewer bits) to symbols with high frequencies;
- **traditional implementation:** when considering characters as symbols, it can comprise text in $\approx 25\%$;

Huffman's Algorithm

- proposed in 1952;
- **main idea:** assign shorter codes (fewer bits) to symbols with high frequencies;
- **traditional implementation:** when considering characters as symbols, it can comprise text in $\approx 25\%$;
- when considering words as symbols, it can comprise text in $\approx 60\%$;

Huffman's Algorithm

- Set of symbols: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;

Huffman's Algorithm

- Set of symbols: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i is frequency of s_i found in the text, for $1 \leq i \leq n$;

Huffman's Algorithm

- Set of symbols: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i is frequency of s_i found in the text, for $1 \leq i \leq n$;
- We want to assign a code to each symbol in order to compress all text;

Huffman's Algorithm

- Set of symbols: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i is frequency of s_i found in the text, for $1 \leq i \leq n$;
- We want to assign a code to each symbol in order to compress all text;
- <u>No code is a prefix of another one</u>: prefix binary tree;

Huffman's Algorithm

- Set of symbols: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i is frequency of s_i found in the text, for $1 \leq i \leq n$;
- We want to assign a code to each symbol in order to compress all text;
- No code is a prefix of another one: prefix binary tree;
- Each symbol s_i is associated to a leaf node in the tree;

Huffman's Algorithm

- Set of symbols: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i is frequency of s_i found in the text, for $1 \leq i \leq n$;
- We want to assign a code to each symbol in order to compress all text;
- No code is a prefix of another one: prefix binary tree;
- Each symbol s_i is associated to a leaf node in the tree;
- Codes for the symbols are binary sequences;

Huffman's Algorithm

- Set of symbols: $S = \{s_1, s_2, \ldots, s_n\}$, $n > 1$;
- f_i is frequency of s_i found in the text, for $1 \leq i \leq n$;
- We want to assign a code to each symbol in order to compress all text;
- No code is a prefix of another one: prefix binary tree;
- Each symbol s_i is associated to a leaf node in the tree;
- Codes for the symbols are binary sequences;
- **Advantage of using prefix codes:** facility for encoding and decoding;

Huffman's Algorithm: encoding and decoding

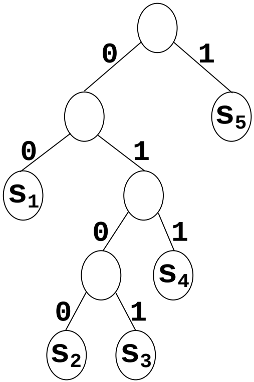
Huffman's Algorithm: encoding and decoding

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

Huffman's Algorithm: encoding and decoding

$S_4 S_3 S_3 S_1 S_3 S_1 S_4 S_5 S_1 S_3 S_3 S_3 S_3 S_3 S_2 S_3 S_5 S_2 S_2 S_2 S_4$

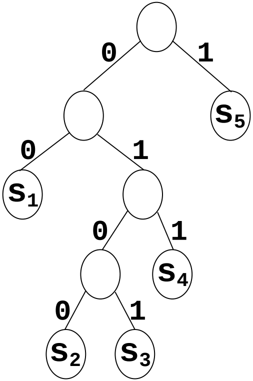
s_1:00
s_2:0100
s_3:0101
s_4:011
s_5:1



Huffman's Algorithm: encoding and decoding

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

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

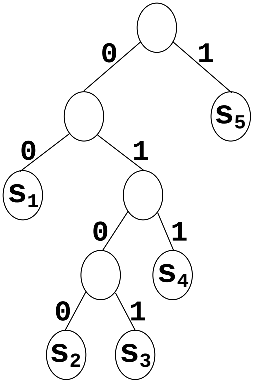


011

Huffman's Algorithm: encoding and decoding

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

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

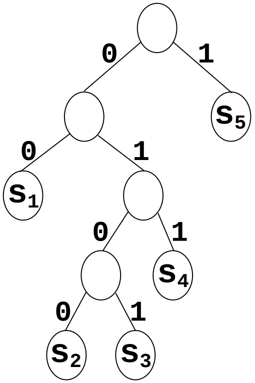


011 0101

Huffman's Algorithm: encoding and decoding

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

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

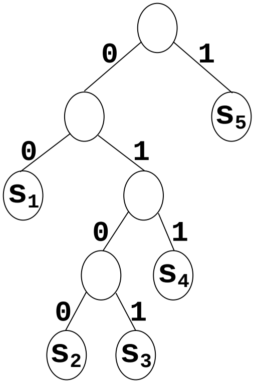


011 0101 0101

Huffman's Algorithm: encoding and decoding

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

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

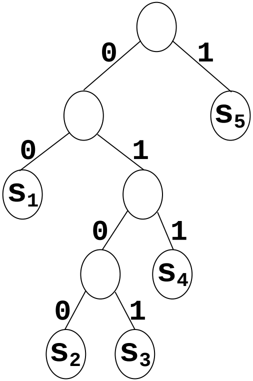


011 0101 0101 00

Huffman's Algorithm: encoding and decoding

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

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



011 0101 0101 00 0101

Huffman's Algorithm: encoding and decoding

$S_4 S_3 S_3 S_1 S_3 S_1 S_4 S_5 S_1 S_3 S_3 S_3 S_3 S_3 S_2 S_3 S_5 S_2 S_2 S_2 S_4$

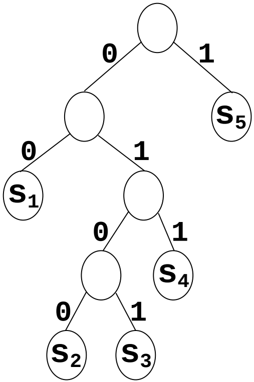
s_1:00

s_2:0100

s_3:0101

s_4:011

s_5:1

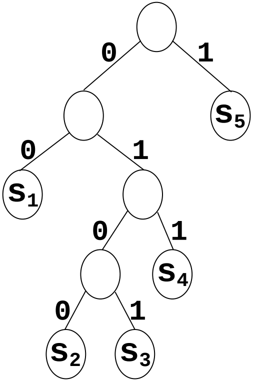


011 0101 0101 00 0101 00

Huffman's Algorithm: encoding and decoding

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

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

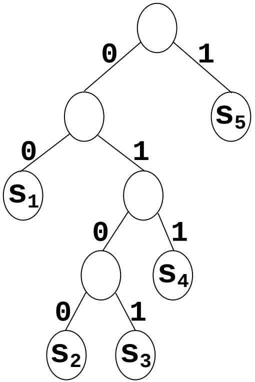


011 0101 0101 00 0101 00 011

Huffman's Algorithm: encoding and decoding

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

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

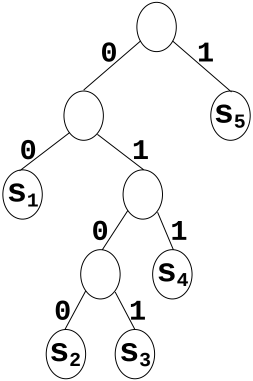


011 0101 0101 00 0101 00 011 1

Huffman's Algorithm: encoding and decoding

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

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

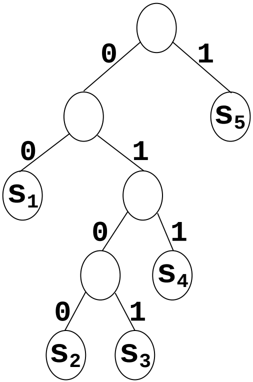


011 0101 0101 00 0101 00 011 1 00

Huffman's Algorithm: encoding and decoding

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

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

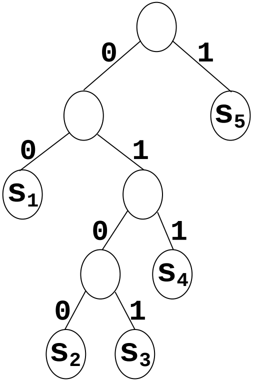


011 0101 0101 00 0101 00 011 1 00 0101

Huffman's Algorithm: encoding and decoding

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

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

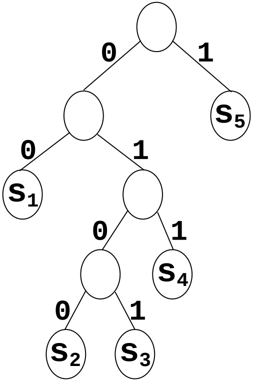


011 0101 0101 00 0101 00 011 1 00 0101 0101

Huffman's Algorithm: encoding and decoding

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

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

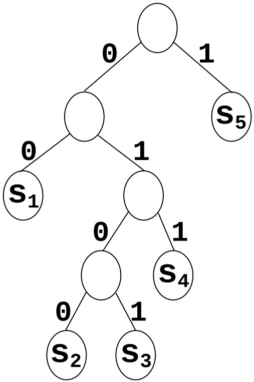


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101

Huffman's Algorithm: encoding and decoding

$S_4 S_3 S_3 S_1 S_3 S_1 S_4 S_5 S_1 S_3 S_3 S_3 S_3 S_3 S_2 S_3 S_5 S_2 S_2 S_2 S_4$

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

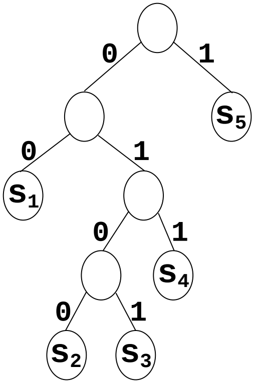


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101

Huffman's Algorithm: encoding and decoding

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

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

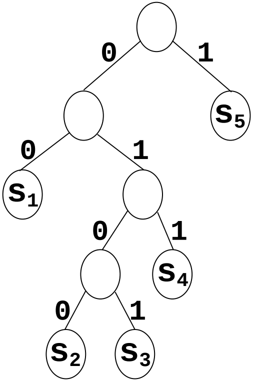


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101 0101

Huffman's Algorithm: encoding and decoding

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

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

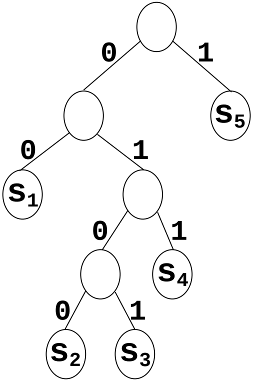


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101 0101 0100

Huffman's Algorithm: encoding and decoding

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

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

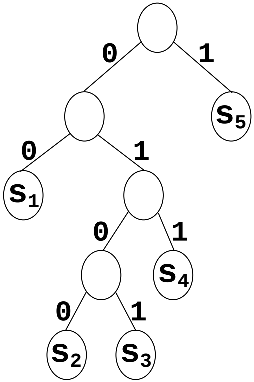


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101 0101 0100 0101

Huffman's Algorithm: encoding and decoding

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

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

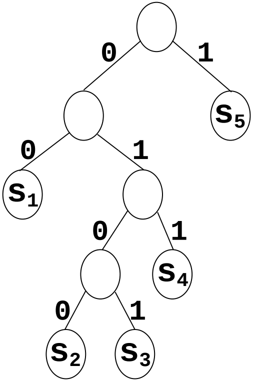


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1

Huffman's Algorithm: encoding and decoding

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

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

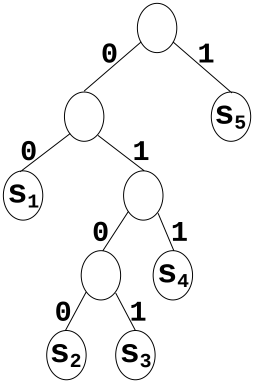


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1 0100

Huffman's Algorithm: encoding and decoding

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

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

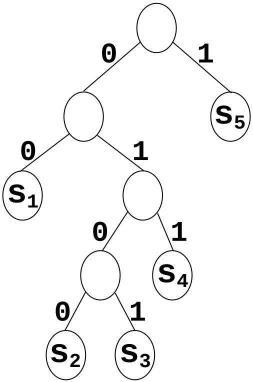


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1 0100 0100

Huffman's Algorithm: encoding and decoding

$S_4 S_3 S_3 S_1 S_3 S_1 S_4 S_5 S_1 S_3 S_3 S_3 S_3 S_3 S_2 S_3 S_5 S_2 S_2 S_2 S_4$

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

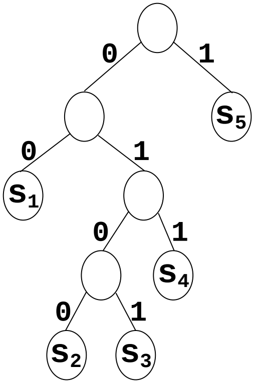


011 0101 0101 00 0101 00 011 1 00 0101 0101 0101 0101
0101 0100 0101 1 0100 0100 0100

Huffman's Algorithm: encoding and decoding

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

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

Huffman's Algorithm

- If prefix tree T is known, encoding/decoding is made in time $O(m)$ where m is the size of the encoded binary sequence;

- If prefix tree T is known, encoding/decoding is made in time $O(m)$ where m is the size of the encoded binary sequence;
- For 0110101010100010100011100010101010101 01010101010001011010001000100011, cost $c(T) = 69$;

Huffman's Algorithm

- If prefix tree T is known, encoding/decoding is made in time $O(m)$ where m is the size of the encoded binary sequence;
- For 0110101010100010100011100010101010101 01010101010001011010001000100011, cost $c(T) = 69$;
- It is necessary to minimize $c(T)$;

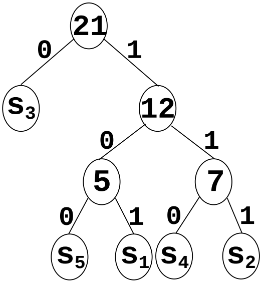
Huffman's Algorithm

- If prefix tree T is known, encoding/decoding is made in time $O(m)$ where m is the size of the encoded binary sequence;
- For 01101010101000101000111000101010101 01010101010001011010001000100011, cost $c(T) = 69$;
- It is necessary to minimize $c(T)$;
- **Minimum Tree** or **Huffman tree**: prefix tree T with the lowest $c(T)$;

Huffman's Algorithm

Huffman's Algorithm

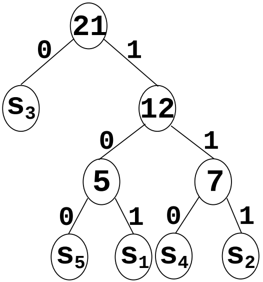
s_1:101
s_2:111
s_3:0
s_4:110
s_5:100



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

Huffman's Algorithm

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

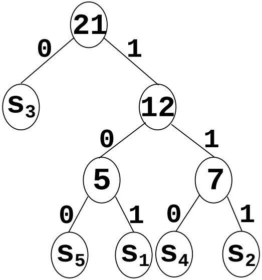


f_1:3
f_2:4
f_3:9
f_4:3
f_5:2

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

Huffman's Algorithm

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

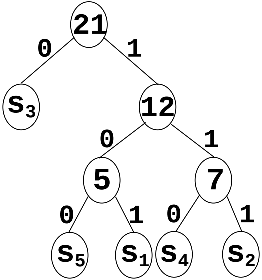


f_1:3
f_2:4
f_3:9
f_4:3
f_5:2

$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$ gives
110001010101110100101000001110100111111111110

Huffman's Algorithm

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

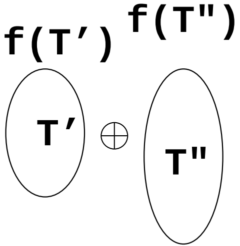


f_1:3
f_2:4
f_3:9
f_4:3
f_5:2

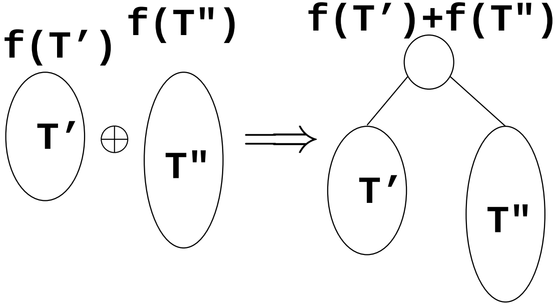
$s_4 s_3 s_3 s_1 s_3 s_1 s_4 s_5 s_1 s_3 s_3 s_3 s_3 s_3 s_2 s_3 s_5 s_2 s_2 s_2 s_4$ gives
110001010101110100101000001110100111111111110
which has $c(T) = 45$ and it is the minimum.

Huffman's Algorithm Idea

Huffman's Algorithm Idea



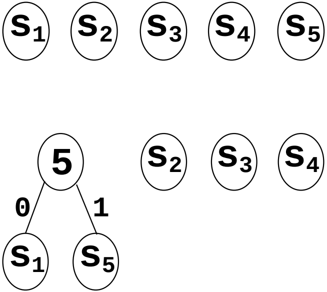
Huffman's Algorithm Idea



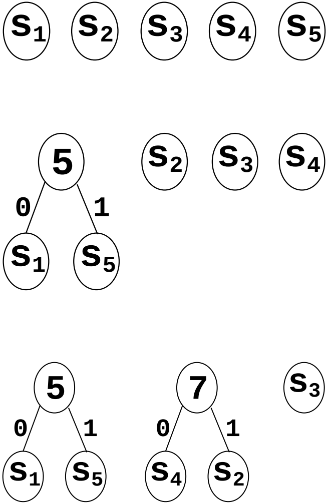
Construction of Huffman Tree

Construction of Huffman Tree

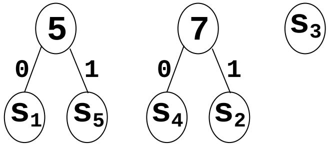
Construction of Huffman Tree



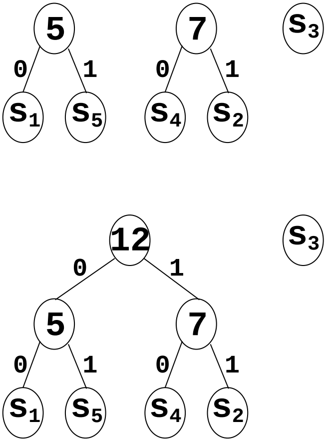
Construction of Huffman Tree



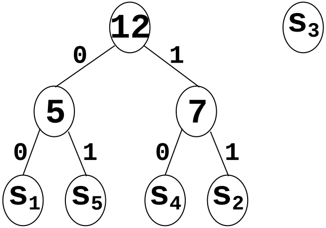
Construction of Huffman Tree



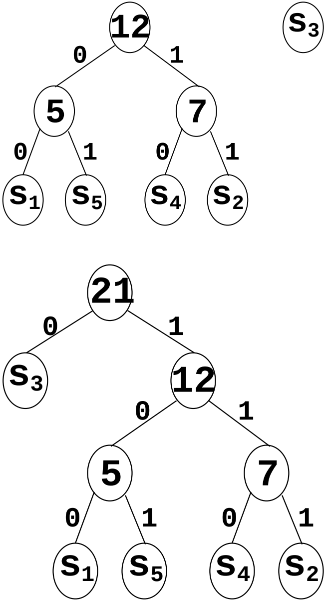
Construction of Huffman Tree



Construction of Huffman Tree



Construction of Huffman Tree



Huffman's Algorithm: inserting into a priority queue

Huffman's Algorithm: inserting into a priority queue

```
insert(T, f, F):
    F[n + 1] ← T
    n++
    ascend(n)
```

Huffman's Algorithm: inserting into a priority queue

```
insert(T, f, F):
    F[n + 1] ← T
    n++
    ascend(n)
```

```
ascend(i):
    j = ⌊i/2⌋
    if j ≥ 1 then
        if T[i] < T[j] then
            T[i] ⇔ T[j]
            ascend(j)
```

Huffman's Algorithm: inserting into a priority queue

insert(T, f, F):
  $F[n+1] \leftarrow T$
  n++
  ascend(n)

ascend(i):
  $j = \lfloor i/2 \rfloor$
  if $j \geq 1$ **then**
    if $T[i] < T[j]$ **then**
      $T[i] \Leftrightarrow T[j]$
      ascend(j)

- **insert(T, f, F) and ascend(i)**: priority queue methods implemented by a heap data structure;

Huffman's Algorithm: deletion from priority queue

Huffman's Algorithm: deletion from priority queue

minimum(T, f, F):
    if $n \neq 0$ **then**
        remove $T[1]$
        $T[1] \leftarrow T[n]$
        n- -
        descend($1, n$)

Huffman's Algorithm: deletion from priority queue

```
minimum(T, f, F):
    if n ≠ 0 then
        remove T[1]
        T[1] ← T[n]
        n- -
        descend(1, n)
```

```
descend(i, n):
    j = 2 ∗ i
    if j ≤ n then
        if j < n then
            if T[j + 1] < T[j] then
                j = j + 1
        if T[i] > T[j] then
            T[i] ⇔ T[j]
            descend(j, n)
```

Huffman's Algorithm: deletion from priority queue

```
minimum(T, f, F):
    if n ≠ 0 then
        remove T[1]
        T[1] ← T[n]
        n- -
        descend(1, n)
```

```
descend(i, n):
    j = 2 ∗ i
    if j ≤ n then
        if j < n then
            if T[j + 1] < T[j] then
                j = j + 1
        if T[i] > T[j] then
            T[i] ⇔ T[j]
            descend(j, n)
```

- **minimum(T, f, F) and descend($1, n$)**: priority queue methods implemented by a heap data structure;

Huffman's Algorithm

huffman():
    for $i \leftarrow 1$ **to** $n-1$ **do**
        minimum(T', f, F);
        minimum(T'', f, F);
        $T \leftarrow T' \oplus T''$;
        $f(T) \leftarrow f(T') \oplus f(T'')$;
        insert(T, f, F);

Huffman's Algorithm

huffman():
    for $i \leftarrow 1$ **to** $n - 1$ **do**
        minimum(T', f, F);
        minimum(T'', f, F);
        $T \leftarrow T' \oplus T''$;
        $f(T) \leftarrow f(T') \oplus f(T'')$;
        insert(T, f, F);

- Operation of minimization or insertion in the priority queue: $O(\log n)$;

Huffman's Algorithm

huffman():
    for $i \leftarrow 1$ **to** $n-1$ **do**
        minimum(T', f, F);
        minimum(T'', f, F);
        $T \leftarrow T' \oplus T''$;
        $f(T) \leftarrow f(T') \oplus f(T'')$;
        insert(T, f, F);

- Operation of minimization or insertion in the priority queue: $O(\log n)$;
- operation $\oplus$ requires constant number of steps;

Huffman's Algorithm

huffman():
  for $i \leftarrow 1$ **to** $n - 1$ **do**
    minimum(T', f, F);
    minimum(T'', f, F);
    $T \leftarrow T' \oplus T''$;
    $f(T) \leftarrow f(T') \oplus f(T'')$;
    insert(T, f, F);

- Operation of minimization or insertion in the priority queue: $O(\log n)$;
- operation $\oplus$ requires constant number of steps;
- Total: $n - 1$ iterations;

Huffman's Algorithm

huffman():
    for $i \leftarrow 1$ **to** $n - 1$ **do**
        minimum(T', f, F);
        minimum(T'', f, F);
        $T \leftarrow T' \oplus T''$;
        $f(T) \leftarrow f(T') \oplus f(T'')$;
        $\mathit{insert}(T, f, F)$;

- Operation of minimization or insertion in the priority queue: $O(\log n)$;
- operation $\oplus$ requires constant number of steps;
- Total: $n - 1$ iterations;
- **Complexity:** $O(n \log n)$;

Exercises

1. Draw a Huffman tree for the following set of keys and frequencies, respectively in the first and second lines of the table:

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. The resulting tree from Huffman's algorithm is unique, i.e., the algorithm always returns the same tree?

Bibliography

SZWARCFITER, J. L. and MARKENZON, L. Estruturas de Dados e seus Algoritmos, LTC, 1994. (in Portuguese)

ZIVIANI, N. Projeto de Algoritmos: com implementações em Java e C++, Cengage Learning, 2009. (in Portuguese)

Questions?