

## 8. Data compression – nonparametric methods

Coding Technology

# Adaptive Huffman codes

Shannon–Fano coding and Huffman coding both use the source distribution. What if this information is not available?

## Adaptive Huffman codes

Shannon–Fano coding and Huffman coding both use the source distribution. What if this information is not available?

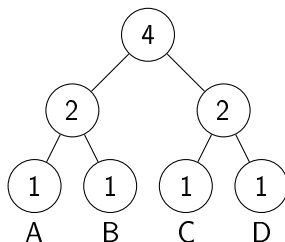
Adaptive Huffman coding: instead of using the source distribution, we build the tree based on the number of characters seen so far in the text. We will start from a tree with 1 occurrence for every character.

# Adaptive Huffman codes

Shannon–Fano coding and Huffman coding both use the source distribution. What if this information is not available?

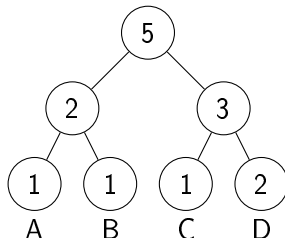
Adaptive Huffman coding: instead of using the source distribution, we build the tree based on the number of characters seen so far in the text. We will start from a tree with 1 occurrence for every character.

Example. If the alphabet is  $\{A,B,C,D\}$ , we initialize the Huffman tree as



## Adaptive Huffman codes

Then the numbers are increased as we read the text. For example, if the first letter of the text is D, we add 1 to the corresponding leaf (and modify the internal nodes accordingly):



And so on for further characters.

## Adaptive Huffman codes – the sibling pair property

We always assume that siblings (two nodes with the same parent) are always ordered increasing from left to right.

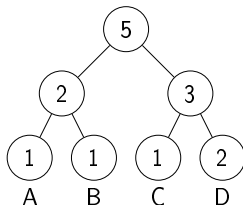
There is one more important property that always needs to hold for Huffman trees: starting from the bottom left, going left to right first, then up one level and repeat, the sequence of numbers obtained has to be non-decreasing. This is called the *sibling pair property*, and trees satisfying this property are *valid Huffman trees*.

## Adaptive Huffman codes – the sibling pair property

We always assume that siblings (two nodes with the same parent) are always ordered increasing from left to right.

There is one more important property that always needs to hold for Huffman trees: starting from the bottom left, going left to right first, then up one level and repeat, the sequence of numbers obtained has to be non-decreasing. This is called the *sibling pair property*, and trees satisfying this property are *valid Huffman trees*.

Example. For the tree



the sequence is 1, 1, 1, 2, 2, 3, 5, which is non-decreasing, so this is a valid Huffman tree.

# Compression

Compression algorithm:

1. Initialize the Huffman tree
2. Read the next character from the text and code it according to the current state of the Huffman tree.
3. Add the next character to the tree.
4. Check the sibling pair property, and if it is violated, restore it by modifying the tree.
5. Move ahead to the next character and repeat from step 2.



# Compression

Compression algorithm:

1. Initialize the Huffman tree
2. Read the next character from the text and code it according to the current state of the Huffman tree.
3. Add the next character to the tree.
4. Check the sibling pair property, and if it is violated, restore it by modifying the tree.
5. Move ahead to the next character and repeat from step 2.

Allowed modifications to restore the sibling pair property:

- ▶ exchange 2 nodes of the same weight (with subtrees), or
- ▶ exchange 2 leaves.

# Decompression

Decompression algorithm:

1. Initialize the Huffman tree
2. Decode 1 character according to the current state of the Huffman tree.
3. Add the decoded character to the tree.
4. Check the sibling pair property, and if it is violated, restore it by modifying the tree.
5. Move ahead to the next character and repeat from step 2.

# Decompression

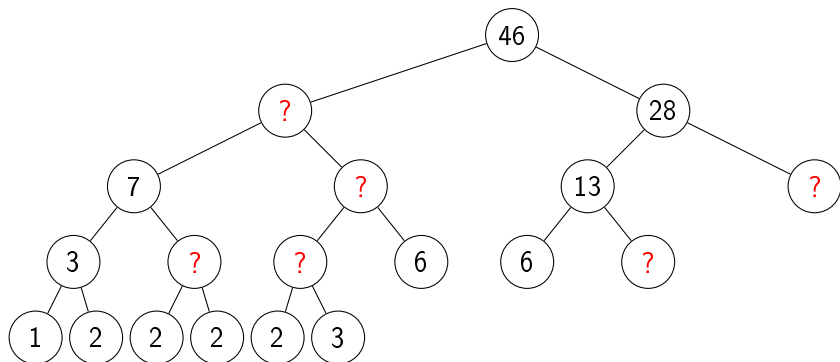
Decompression algorithm:

1. Initialize the Huffman tree
2. Decode 1 character according to the current state of the Huffman tree.
3. Add the decoded character to the tree.
4. Check the sibling pair property, and if it is violated, restore it by modifying the tree.
5. Move ahead to the next character and repeat from step 2.

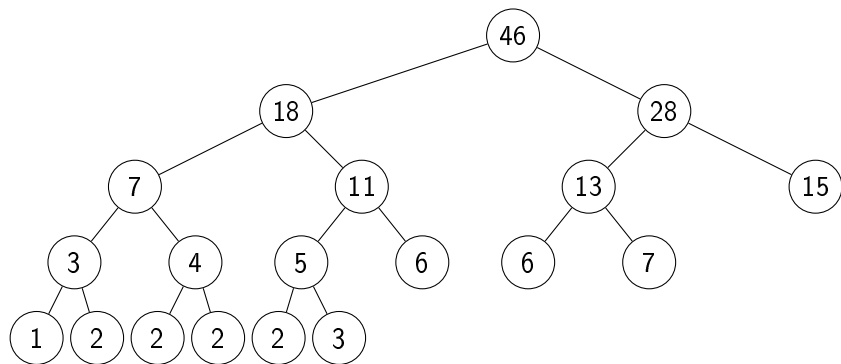
(Step 1 has to be carried out identically during compression and decompression. Step 4 too.)

## Problem 1

Determine the missing values. Does the graph represent a valid Huffman tree?



## Problem 1



Starting from the bottom left, going left to right first, then up one level and repeat, the sequence 1, 2, 2, 2, 2, 3, 3, 4, 5, 6, 6, 7, 11, 13, 15, 18, 28, 46 is non-decreasing, so this is a valid Huffman-tree.

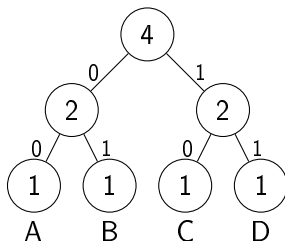
## Problem 2

For the alphabet  $\{A, B, C, D\}$ , depict the progress of the code-tree for the adaptive Huffman-code when encoding the sequence DCDADD.

## Problem 2

For the alphabet  $\{A, B, C, D\}$ , depict the progress of the code-tree for the adaptive Huffman-code when encoding the sequence DCDADD.

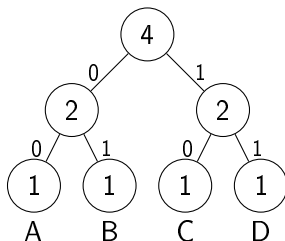
Solution.



## Problem 2

For the alphabet  $\{A, B, C, D\}$ , depict the progress of the code-tree for the adaptive Huffman-code when encoding the sequence DCDADD.

Solution.



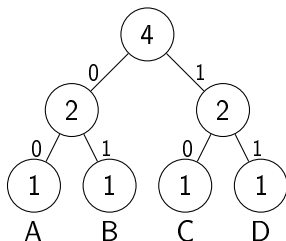
D  $\rightarrow$  11



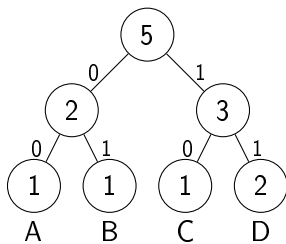
## Problem 2

For the alphabet  $\{A, B, C, D\}$ , depict the progress of the code-tree for the adaptive Huffman-code when encoding the sequence DCDADD.

Solution.



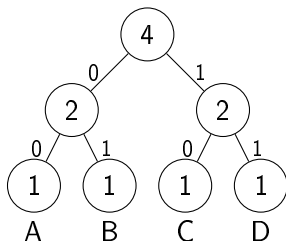
D → 11



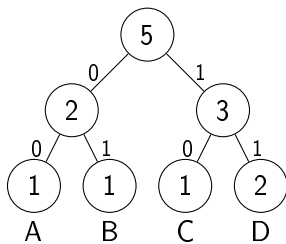
## Problem 2

For the alphabet  $\{A, B, C, D\}$ , depict the progress of the code-tree for the adaptive Huffman-code when encoding the sequence DCDADD.

Solution.

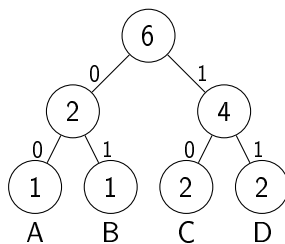


D → 11



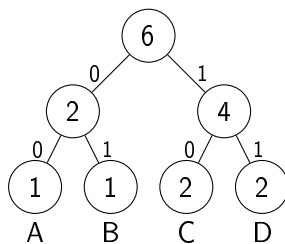
DC → 1110

## Problem 2

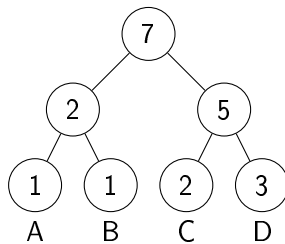


DCD  $\rightarrow$  111011

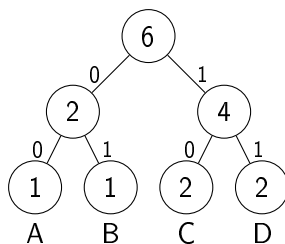
## Problem 2



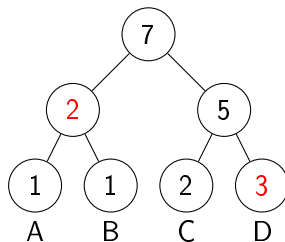
DCD  $\rightarrow$  111011



## Problem 2



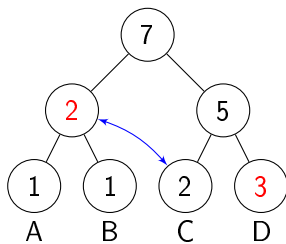
DCD  $\rightarrow$  111011



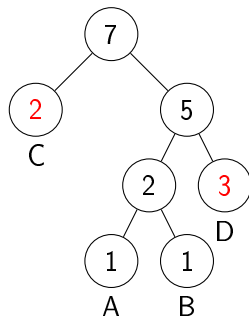
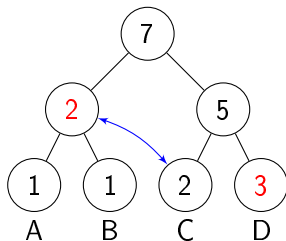
Possible moves to restore the sibling pair property:

- ▶ exchange 2 nodes of the same weight (with subtrees), or
- ▶ exchange 2 leaves.

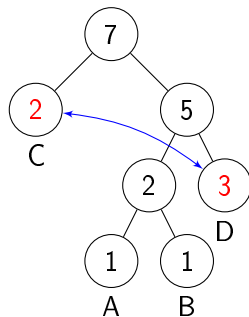
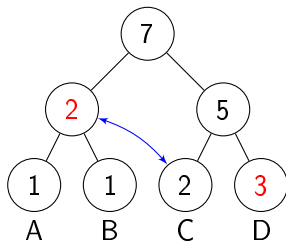
## Problem 2



## Problem 2

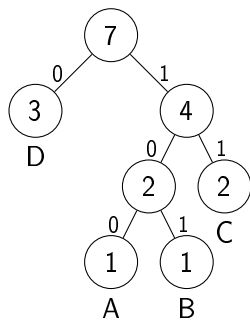


## Problem 2

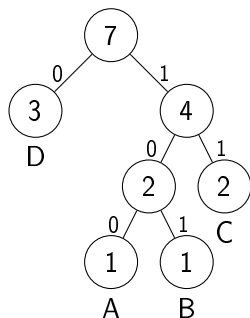




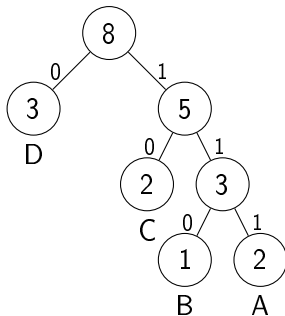
## Problem 2



## Problem 2

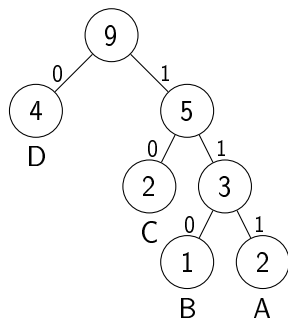


DCD**A**  $\rightarrow$  111011**100**



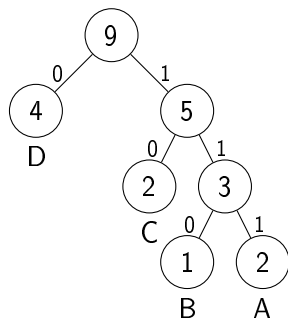
DCDA**D**  $\rightarrow$  111011100**0**

## Problem 2

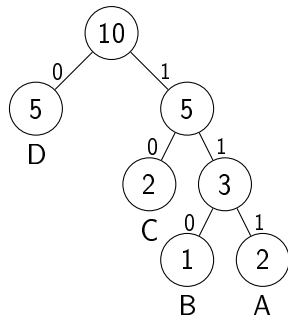


DCDAD $\color{red}{D}$   $\rightarrow$  1110111000 $\color{red}{0}$

## Problem 2

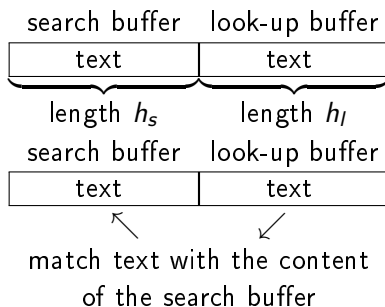


DCDAD**D**  $\rightarrow$  1110111000**0**



## LZ77 algorithm

Main idea: looking ahead, we search for sections of text that were seen recently.



Output:  $(p, l, n)$ , where:

- ▶  $p$ : starting position of the matching section in the search buffer (backwards from the cursor position)
- ▶  $l$ : length of the matching section
- ▶  $n$ : code of the next symbol

Total size of output:  $\lceil \log_2 h_s \rceil + \lceil \log_2 h_l \rceil + \lceil \log_2 \chi \rceil$  bits.

## Problem 3

We use LZ77 to compress the text “c a b r a c a d a b r a r r a d . . .” with parameters  $h_s = 7$ ,  $h_l = 6$ . The cursor is initially at position 7. Give the output of the compression algorithm for the initial state and the next two steps.

## Problem 3

We use LZ77 to compress the text “c a b r a c a d a b r a r r a d . . .” with parameters  $h_s = 7$ ,  $h_l = 6$ . The cursor is initially at position 7. Give the output of the compression algorithm for the initial state and the next two steps.

Solution.

- ▶ initial state:

c a b r a c a	d a b r a r
---------------	-------------

 r a r r a d . . .    output:  $(0, 0, d)$

### Problem 3


We use LZ77 to compress the text “c a b r a c a d a b r a r r a d . . .” with parameters  $h_s = 7, h_l = 6$ . The cursor is initially at position 7. Give the output of the compression algorithm for the initial state and the next two steps.

**Solution.**

- ▶ initial state:

cabraca	dabrar	rarrad...	output: $(0, 0, d)$
---------	--------	-----------	---------------------

- next step:


 c abra cad abra rrad...    output: (7, 4, r)



### Problem 3


We use LZ77 to compress the text “c a b r a c a d a b r a r r a d . . .” with parameters  $h_s = 7, h_l = 6$ . The cursor is initially at position 7. Give the output of the compression algorithm for the initial state and the next two steps.

**Solution.**

- ▶ initial state:

cabraca	dabrar	rarrad...	output: $(0, 0, d)$
---------	--------	-----------	---------------------

- ▶ next step:


 c abra cad abra rr arrad...    output: (7, 4, r)

- ▶ next step:

ca br ac ad ab rar rarra d... output: (3, 5, d)

## Problem 4

We use LZ77 data compressor for a text written in an alphabet of size 32. The search buffer length is 32 and the look-up buffer length is 16. How many bits is the length of the output sequence at each step of the compression?

## Problem 4

We use LZ77 data compressor for a text written in an alphabet of size 32. The search buffer length is 32 and the look-up buffer length is 16. How many bits is the length of the output sequence at each step of the compression?

Solution.

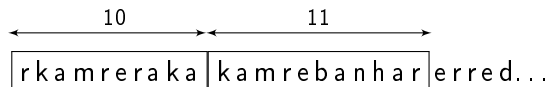
$$\chi = 32, \quad h_s = 32, \quad h_l = 16,$$

so

$$\lceil \log_2 h_s \rceil + \lceil \log_2 h_l \rceil + \lceil \log_2 \chi \rceil = 5 + 4 + 5 \text{ bits.}$$

## Problem 5

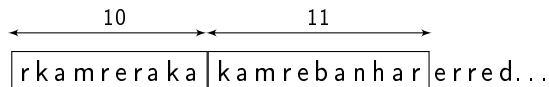
Running data compression by LZ77, the shift register is in the state



Give the output of the next step.

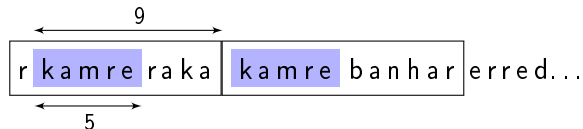
## Problem 5

Running data compression by LZ77, the shift register is in the state



Give the output of the next step.

Solution.



output:  $(9, 5, b)$

# LZ78 algorithm

Main idea: we are parsing the text into sections that are 1 character longer than a section seen before. The output (coding for the new section) is  $(i, c)$ , where

- ▶  $i$  is the address of the old section (1 character shorter than the current section),
- ▶  $c$  is the new character (novelty factor).

The address of the current section is incremented by 1 for each new section.

# LZ78 algorithm

Example.

a b a b b b b b a b b a b


# LZ78 algorithm

Example.

a b a b b b b b a b b a b

Coding.

1. "a" is a new section, so the output is (0, a).

1	(0,a)



# LZ78 algorithm

Example.

a b a b b b b b a b b a b

Coding.

1. "a" is a new section, so the output is (0, a).
2. "b" is a new section, so the output is (0, b).

1	(0,a)
2	(0,b)

# LZ78 algorithm

Example.

a b a b b b b b a b b a b

Coding.

1. "a" is a new section, so the output is (0, a).
2. "b" is a new section, so the output is (0, b).
3. "a" is an old section with address 1; the new section is "ab", the new character is "b", so the output is (1, b).

1	(0,a)
2	(0,b)
3	(1,b)

# LZ78 algorithm

Example.

a b a b b b b b a b b a b

Coding.

1. "a" is a new section, so the output is (0, a).
2. "b" is a new section, so the output is (0, b).
3. "a" is an old section with address 1; the new section is "ab", the new character is "b", so the output is (1, b).
4. "b" is an old section with address 2; the new section is "bb", and the new character is "b", so the output is (2, b).

1	(0,a)
2	(0,b)
3	(1,b)
4	(2,b)

# LZ78 algorithm

Example.

a b a b b b b b a b b a b

Coding.

1. "a" is a new section, so the output is (0, a).
2. "b" is a new section, so the output is (0, b).
3. "a" is an old section with address 1; the new section is "ab", the new character is "b", so the output is (1, b).
4. "b" is an old section with address 2; the new section is "bb", and the new character is "b", so the output is (2, b).
5. "bb" is an old section with address 4; the new section is "bba", and the new character is "a", so the output is (4, a).

1	(0,a)
2	(0,b)
3	(1,b)
4	(2,b)
5	(4,a)

# LZ78 algorithm

Example.

a b a b b b b b a b b a b

Coding.

1. "a" is a new section, so the output is (0, a).
2. "b" is a new section, so the output is (0, b).
3. "a" is an old section with address 1; the new section is "ab", the new character is "b", so the output is (1, b).
4. "b" is an old section with address 2; the new section is "bb", and the new character is "b", so the output is (2, b).
5. "bb" is an old section with address 4; the new section is "bba", and the new character is "a", so the output is (4, a).
6. "bba" is an old section with address 5; the new section is "bbab" and the new character is "b", so the output is (5, b).

1	(0,a)
2	(0,b)
3	(1,b)
4	(2,b)
5	(4,a)
6	(5,b)

## Problem 6

Compress the sequence 01000101001010001101011 using the LZ78 algorithm. Use binary addresses.

## Problem 6

Compress the sequence 01000101001010001101011 using the LZ78 algorithm. Use binary addresses.

Solution.

section	0	1	00	01	010	0101	000	11	01011
address	1	2	3	4	5	6	7	8	9
bin. addr.	0001	0010	0011	0100	0101	0110	0111	1000	1001

The compressed sequence is

(0000,0) (0000,1) (0001,0) (0001,1) (0100,0) (1001,1) (0011,0)  
(0010,1) (0110,1)

## Problem 7

Decompress the sequence 0001000000100110100110101101 using the LZ78 algorithm.



## Problem 7

Decompress the sequence 0001000000100110100110101101 using the LZ78 algorithm.

Solution. Step 0. How many bits is the address length?

## Problem 7

Decompress the sequence 0001000000100110100110101101 using the LZ78 algorithm.

Solution. Step 0. How many bits is the address length?

It cannot be 2, because with 2 bit addresses, at most 4 sections can be coded, and the total length of the coded text could be at most 12.

## Problem 7

Decompress the sequence 0001000000100110100110101101 using the LZ78 algorithm.

Solution. Step 0. How many bits is the address length?

It cannot be 2, because with 2 bit addresses, at most 4 sections can be coded, and the total length of the coded text could be at most 12. It cannot be 4 or longer, because the first address is all 0's. So the address length must be 3 bits.

## Problem 7

Decompress the sequence 0001000000100110100110101101 using the LZ78 algorithm.

Solution. Step 0. How many bits is the address length?

It cannot be 2, because with 2 bit addresses, at most 4 sections can be coded, and the total length of the coded text could be at most 12. It cannot be 4 or longer, because the first address is all 0's. So the address length must be 3 bits.

Step 1. Decomposing to addresses and novelty factors:

$(000, 1) (000, 0) (001, 0) (011, 0) (100, 1) (101, 0) (110, 1)$

Step 2. Reconstructing the original text:

1 0 10 100 1001 10010 100101