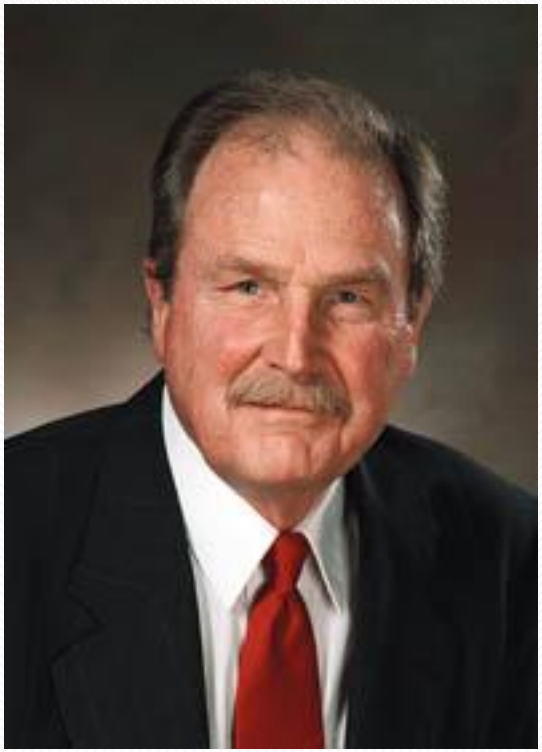
Huffman coding algorithm

Andrzej Pisarski

Plan of the seminar

- David Huffman
- Aplications of Huffman algorithm
- Huffman algorithm
  - Concepts
  - Compression (encoding)
  - Unpacking (decoding)

David Huffman



- Born on 9 august 1925 – died on 7 october 1999 at Santa Cruz, California.
- Came from Ohio and was educated at a local university. After World War II, he returned to Ohio State University to earn a Master's degree in electrical engineering (1949). In 1953 he earned his Doctor of Science in electrical engineering at the Massachusetts Institute of Technology. He later worked as a lecturer at MIT, and from 1967 at the University of California in Santa Cruz.

http://en.wikipedia.org/wiki/David_A._Huffman

David Huffman

- Huffman did not patent his achievements, focusing on educational work. He used to say: 'my products are students'.

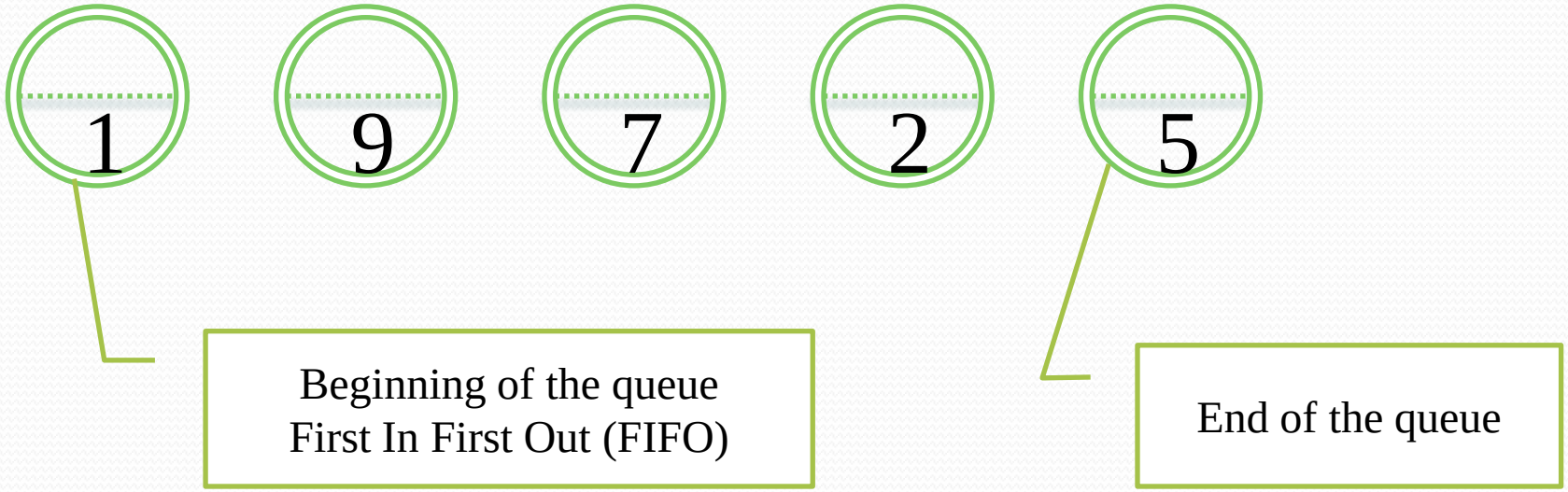
http://pl.wikipedia.org/wiki/David_Huffman

Application of Huffman coding

- „Huffman is probably best known for the development of the Huffman Coding Procedure, the result of a term paper he wrote while a graduate student at the Massachusetts Institute of Technology (MIT). "Huffman Codes" are used in nearly every application that involves the compression and transmission of digital data, such as fax machines, modems, computer networks, and high-definition television."
- Compression of audio files MP3

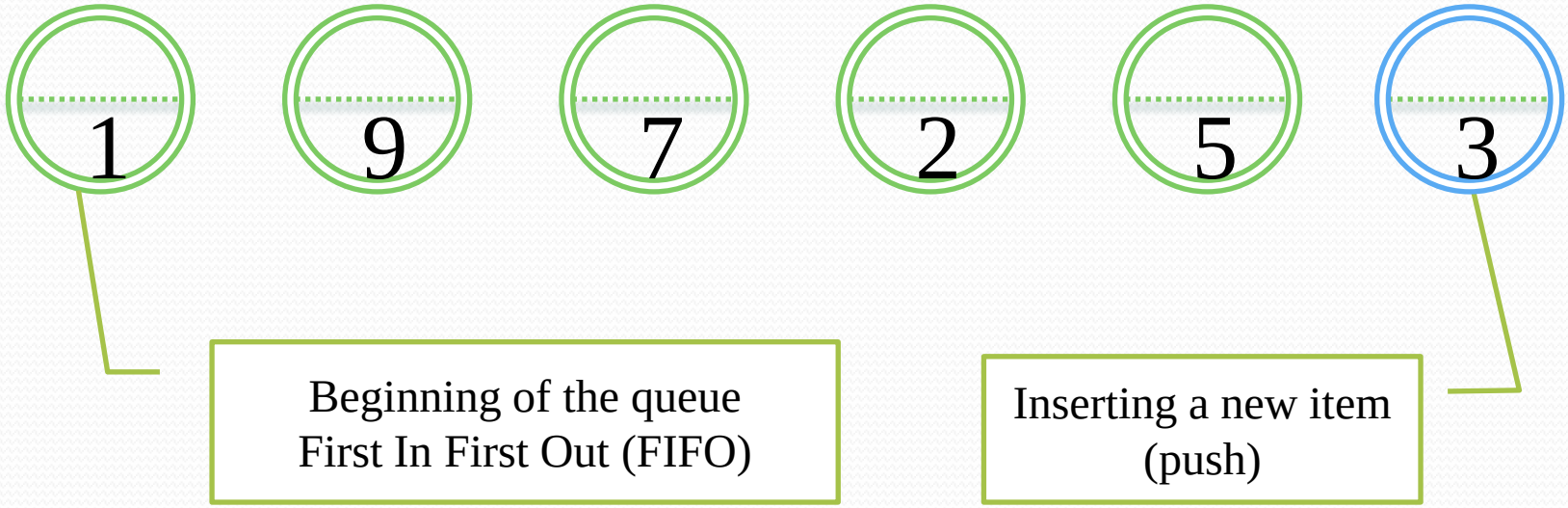
  http://www1.ucsc.edu/currents/99-00/10-11/huffman.html

Huffman algorithm: concepts
Data structure: queue

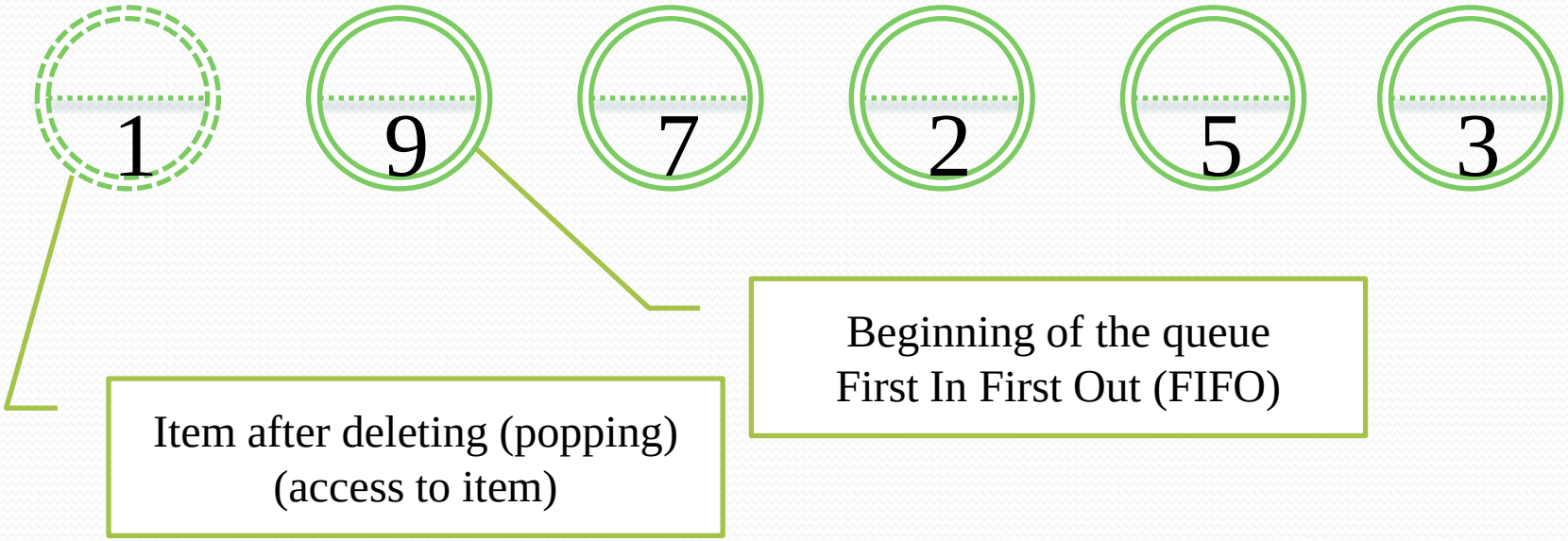


Huffman algorithm: concepts

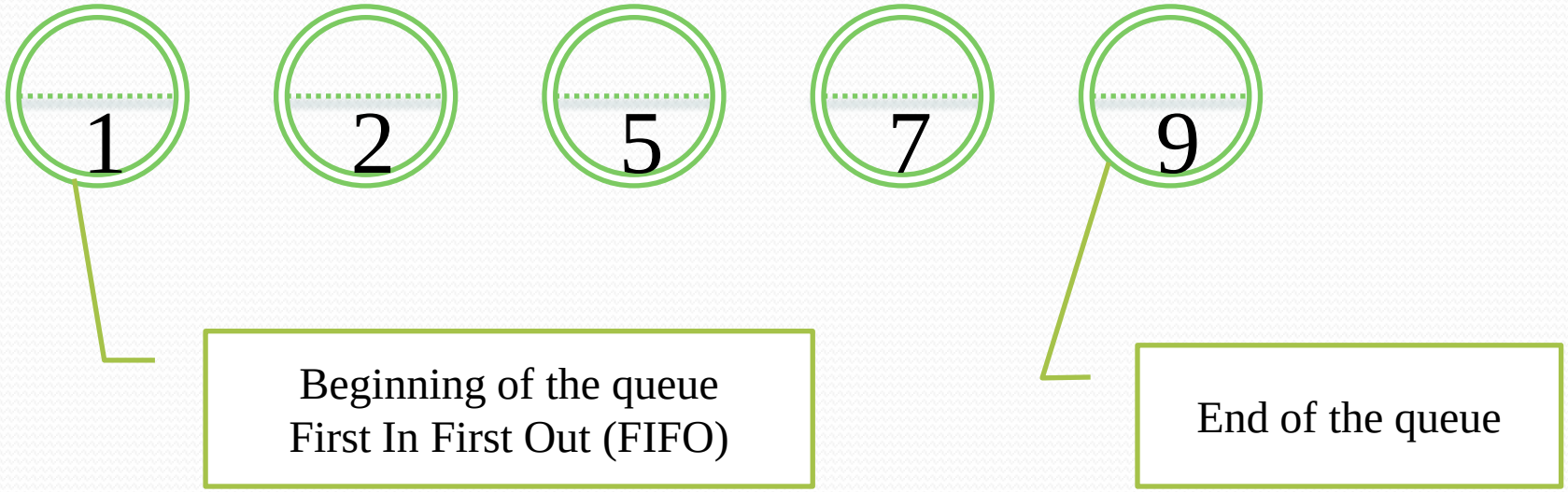
Data structure: queue



Huffman algorithm: concepts
Data structure: queue

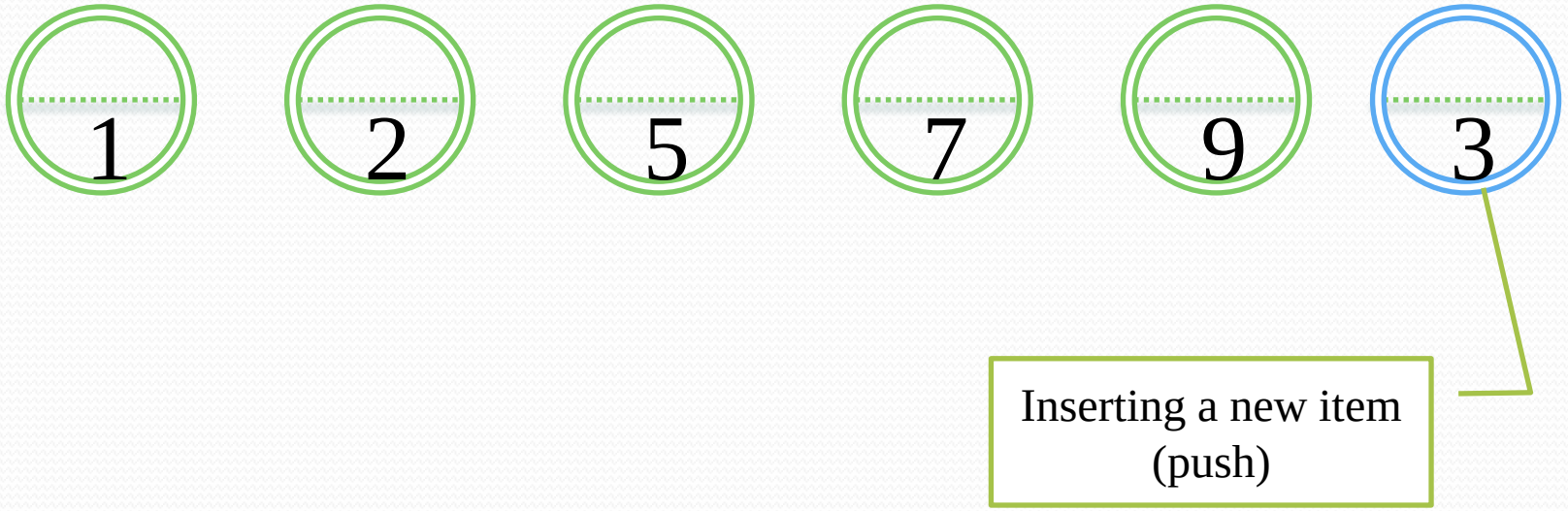


Huffman algorithm: concepts
Data structure: priority queue



Huffman algorithm: concepts

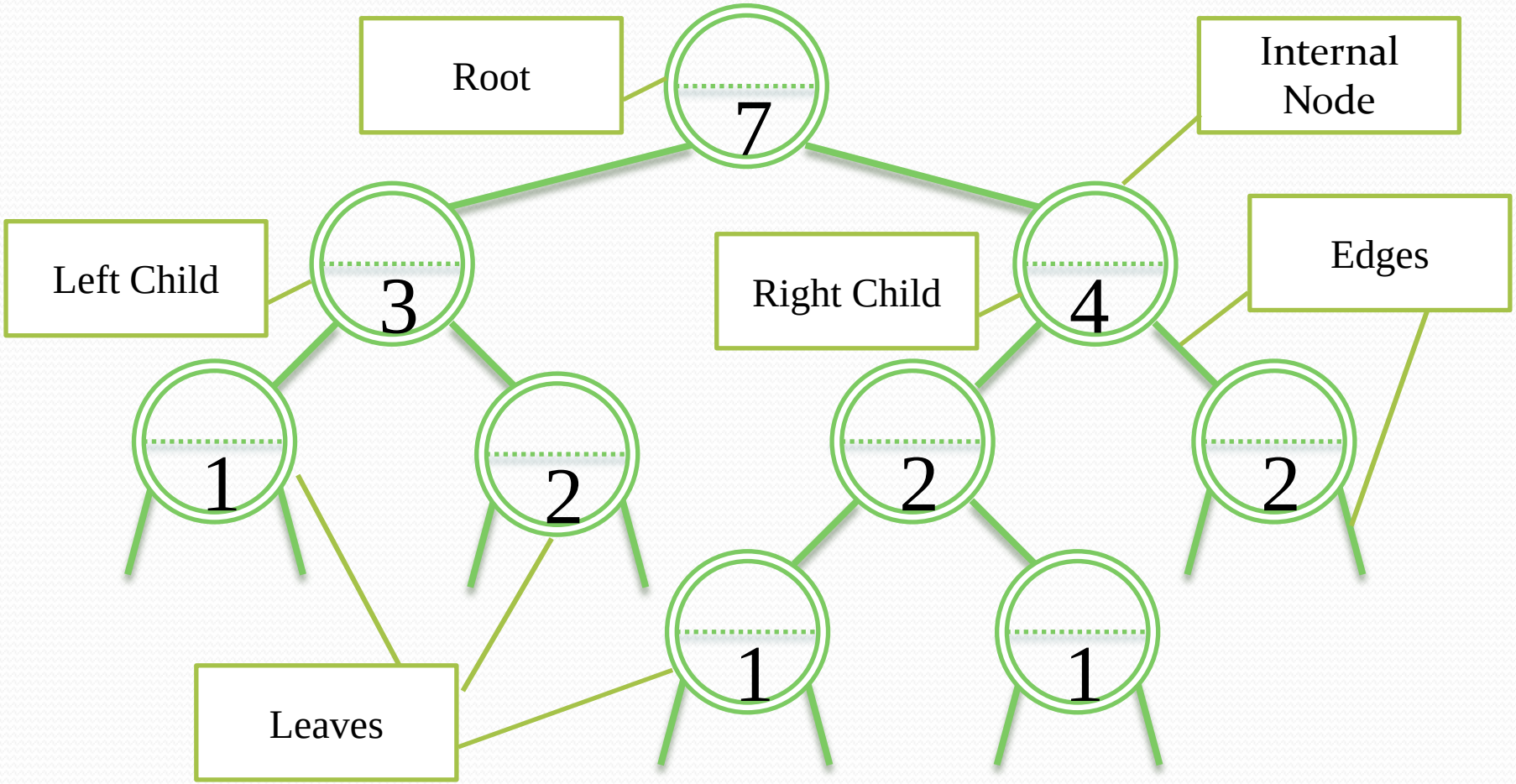
Data structure: priority queue



Huffman algorithm: concepts

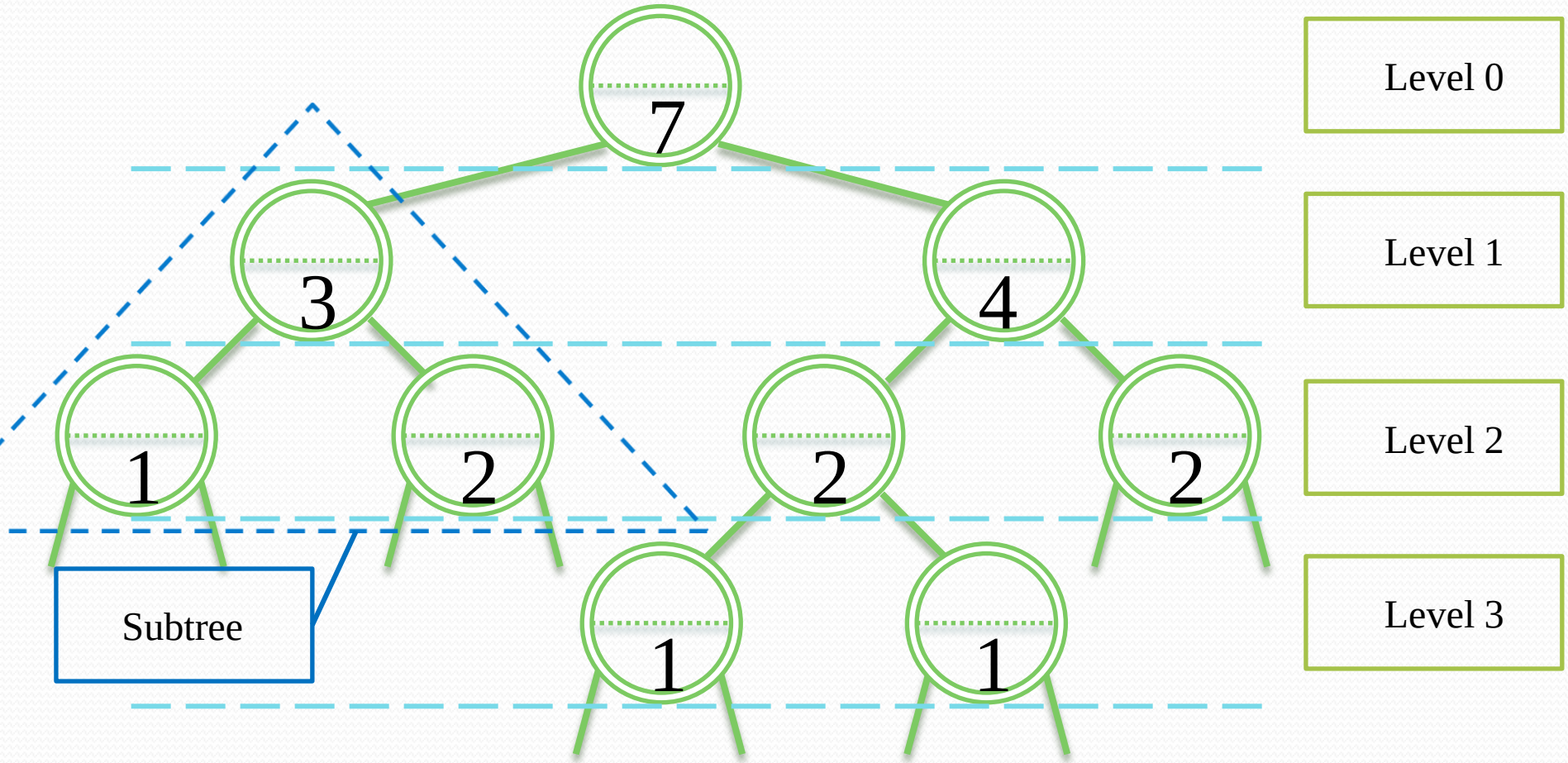
Data structure: priority queue

Huffman algorithm: concepts
Data structure: tree



Huffman algorithm: concepts

Data structure: tree



Huffman algorithm: encoding

„ABRACADABRA!”

SIGN	CODEWORDS		
	Code page ANSI ASCII	PREFIX-FREE CODE	PREFIX CODE
!	00100001	1100	100
A	01000001	0	0
B	01000010	101	1
C	01000011	1101	00
D	01000100	100	10
R	01010010	111	11
B [b]	**12 [96]**	**4 [28+4]**	**3 [18+6]**

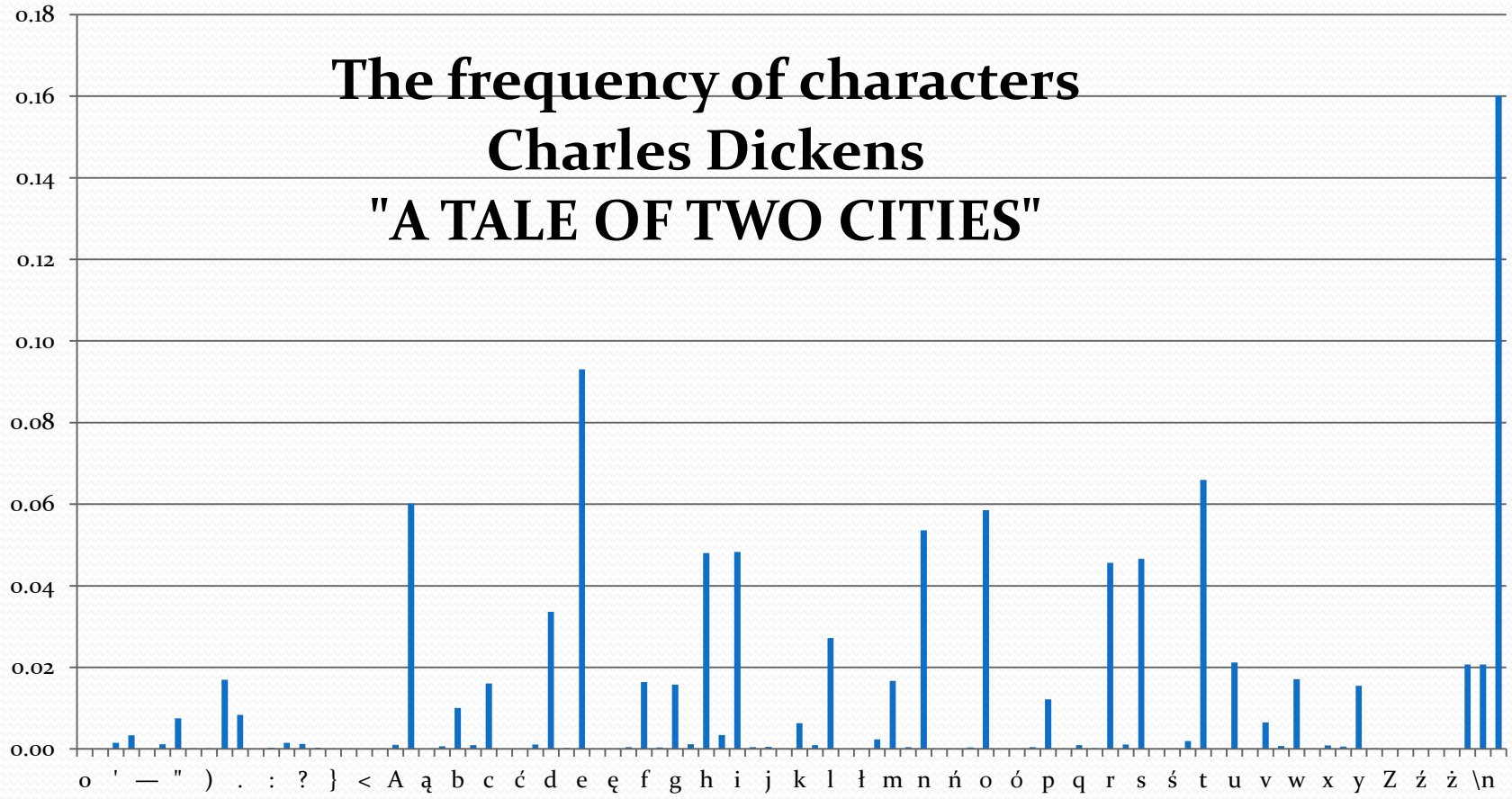
Huffman algorithm: encoding (1/7) Read data from an input

1. Read data from an input.
2. Store in array frequency of signs.
3. Create a Huffman tree.
4. Create an array of codewords.
5. Write the Huffman tree as a stream of bits.
6. Write the number of input characters .
7. Write prefix-free codewords (translated characters from the input).

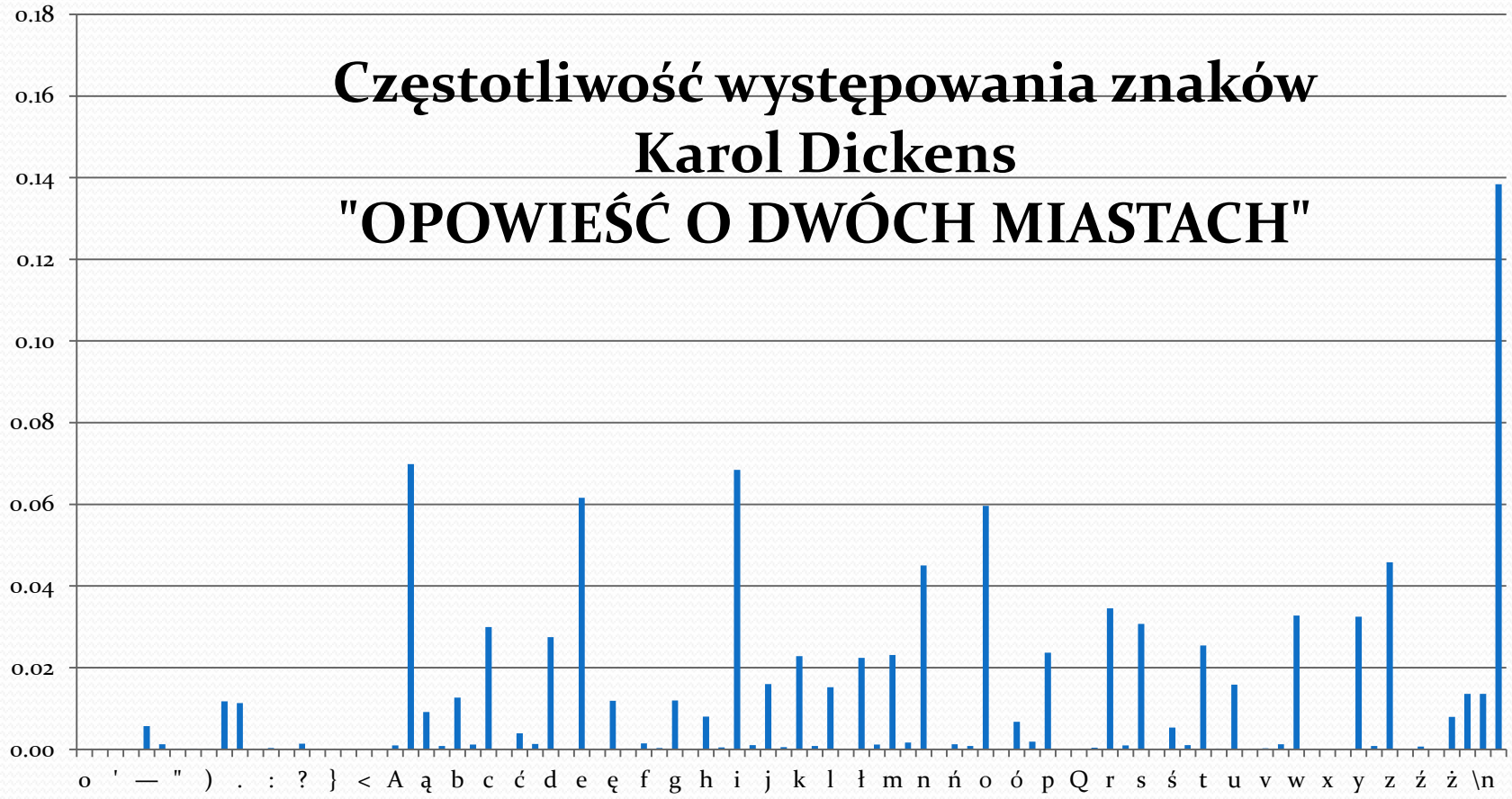
Huffman algorithm: encoding (2/7) frequency of signs

1. Read data from an input.
2. Store in array frequency of signs.
3. Create a Huffman tree.
4. Create an array of codewords.
5. Write the Huffman tree as a stream of bits.
6. Write the number of input characters .
7. Write prefix-free codewords (translated characters from the input).

Huffman algorithm: encoding (2/7) frequency of signs



Huffman algorithm: encoding (2/7) frequency of signs



Huffman algorithm: encoding (2/7) frequency of signs (array)

A TALE OF TWO CITIES	
SPACE	0,1602
e	0,09305
t	0,06591
a	0,06017
o	0,05856
n	0,05356
i	0,04828
h	0,048
s	0,04659
r	0,04562
…	
V	0,00013
X	0,00010
K	0,00008
U	0,00004
Q	0,00004

OPOWIEŚĆ O DWÓCH MIASTACH	
SPACJA	0.1384
a	0.06983
i	0.06841
e	0.06158
o	0.05964
z	0.04577
n	0.04504
r	0.03451
w	0.03275
y	0.03247
…	
q	2.5e-006
}	2.5e-006
/	2.5e-006
Ń	2.5e-006
<	2.5e-006

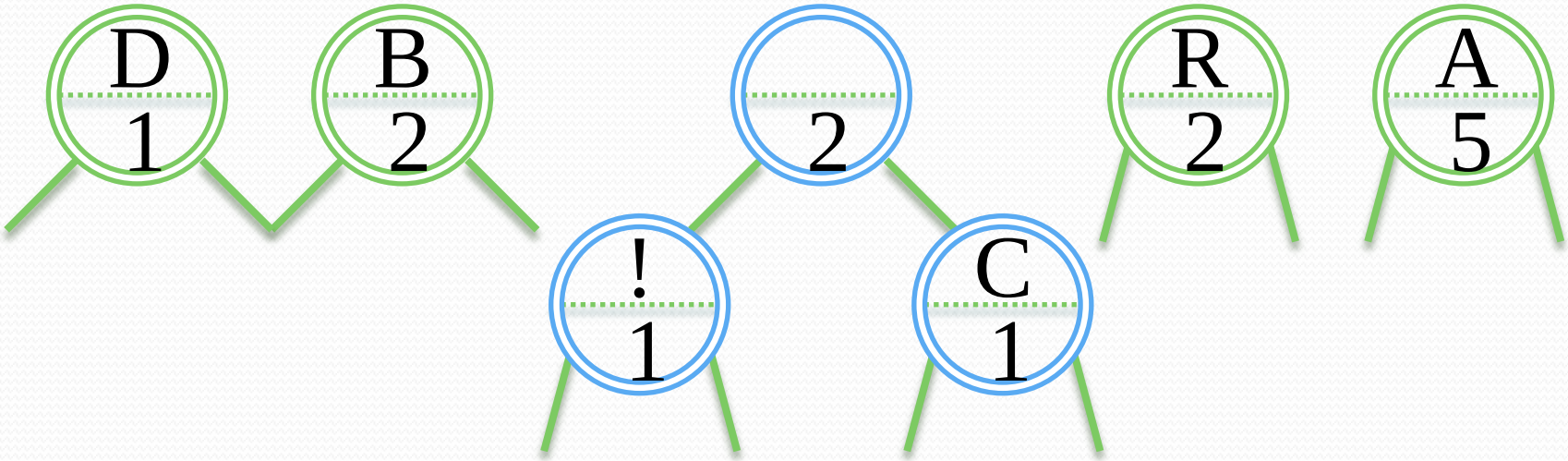
Huffman algorithm: encoding
(3/7) Huffman coding tree

„ABRACADABRA!”

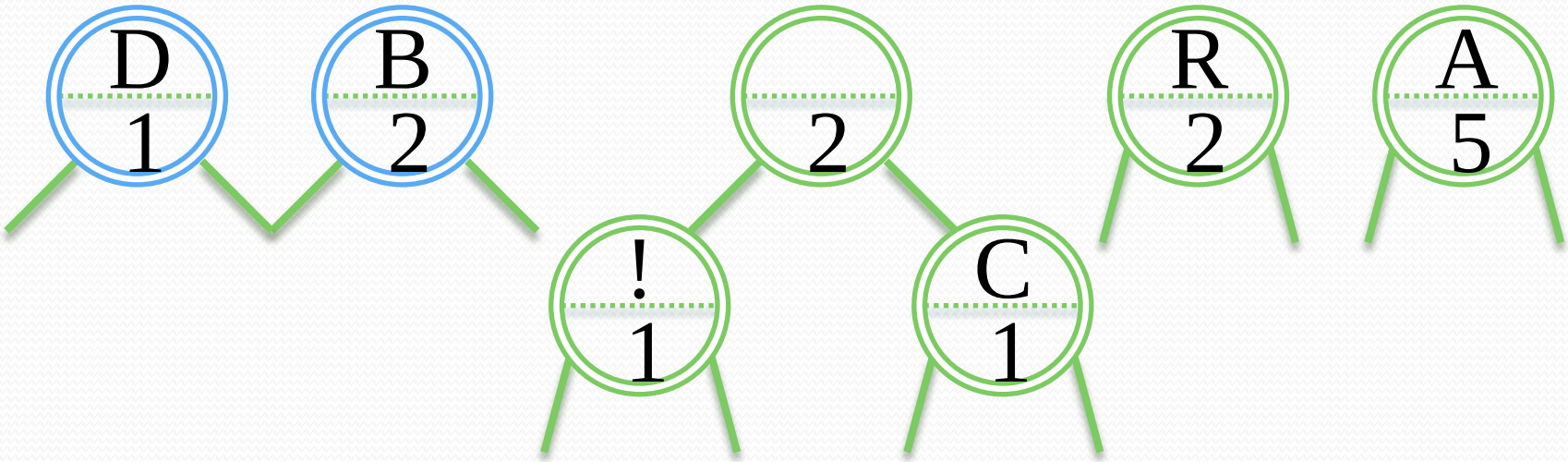
Huffman algorithm: encoding
(3/7) Huffman coding tree

„ABRACADABRA!”

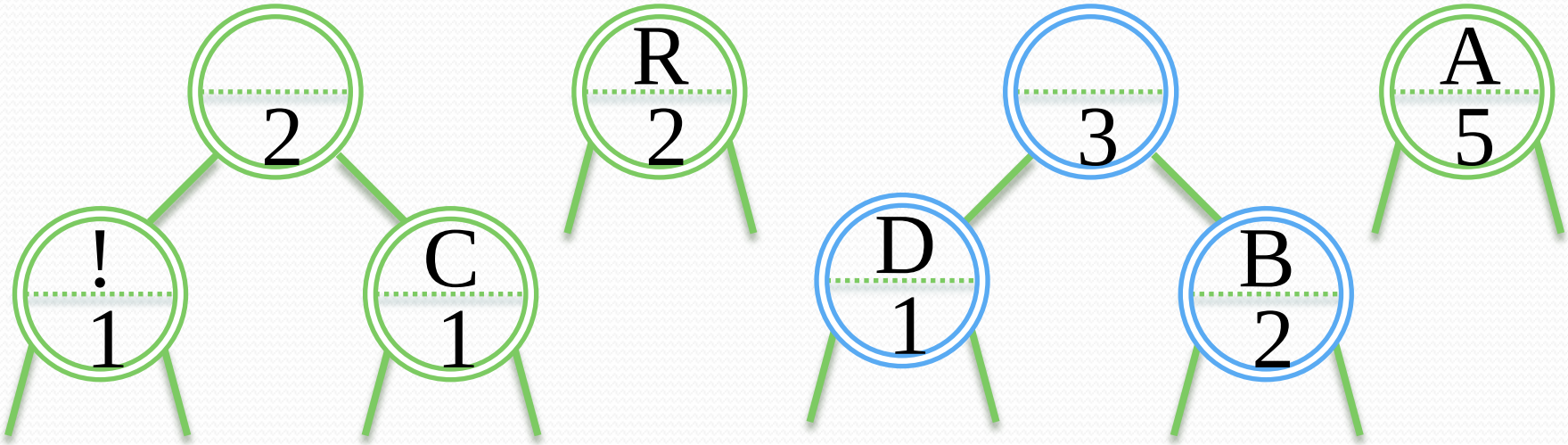
Huffman algorithm: encoding
(3/7) Huffman coding tree



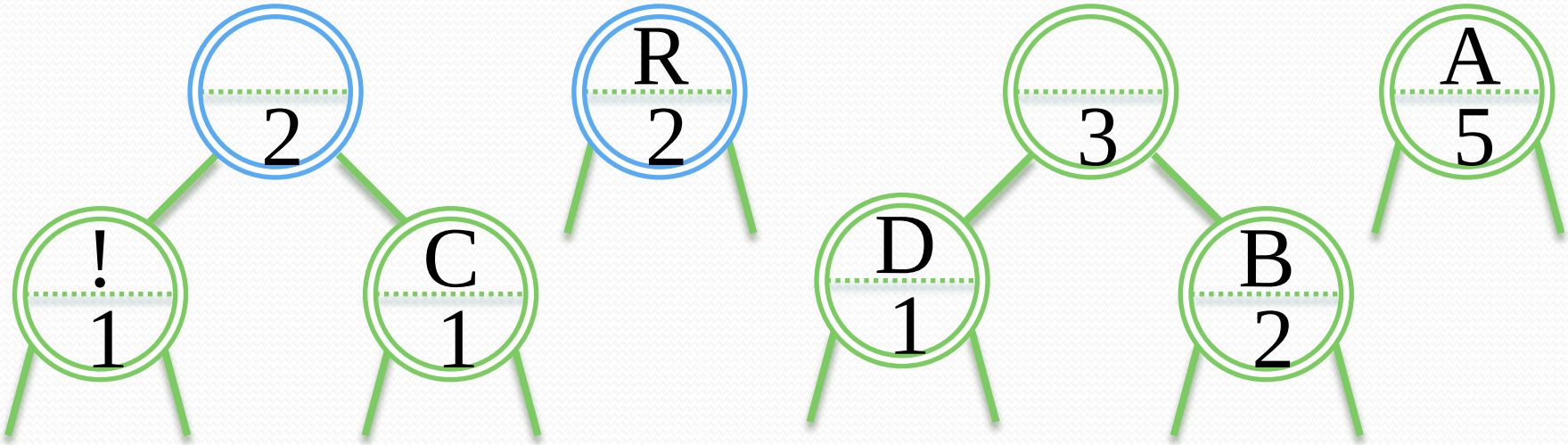
Huffman algorithm: encoding
(3/7) Huffman coding tree



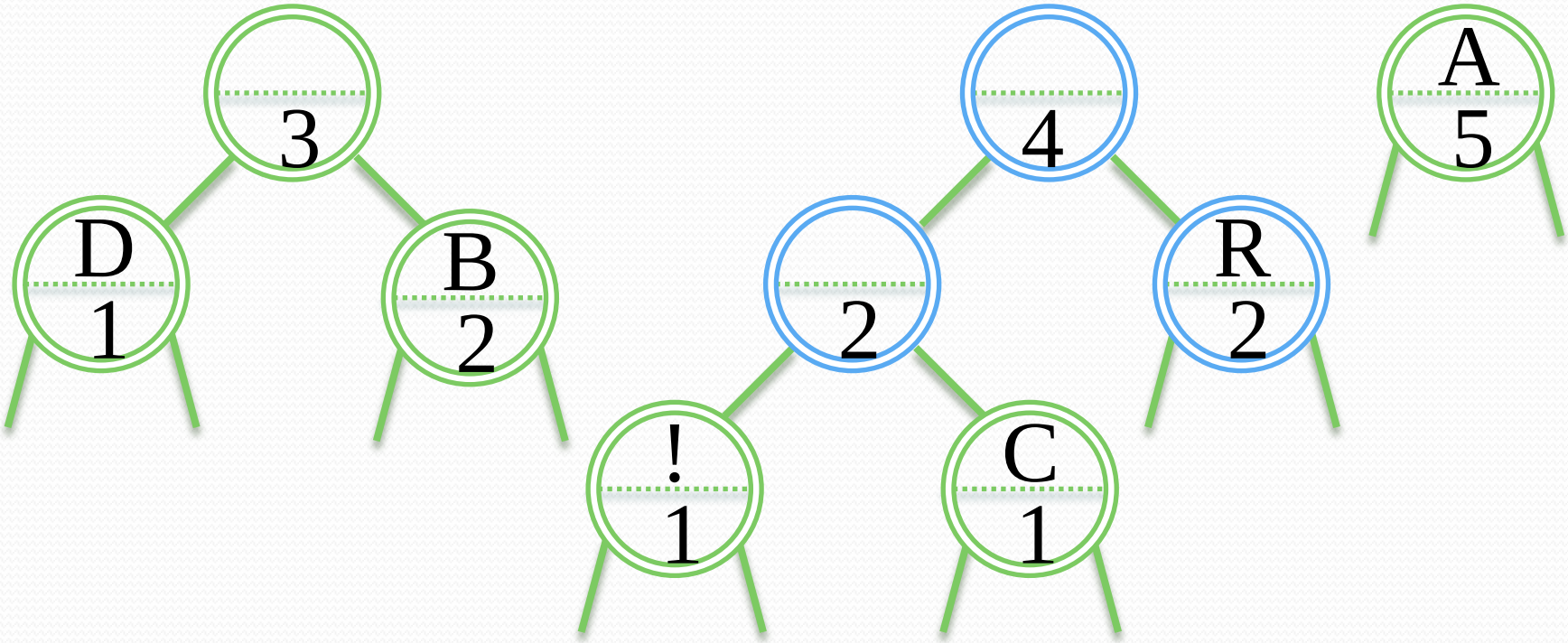
Huffman algorithm: encoding (3/7) Huffman coding tree



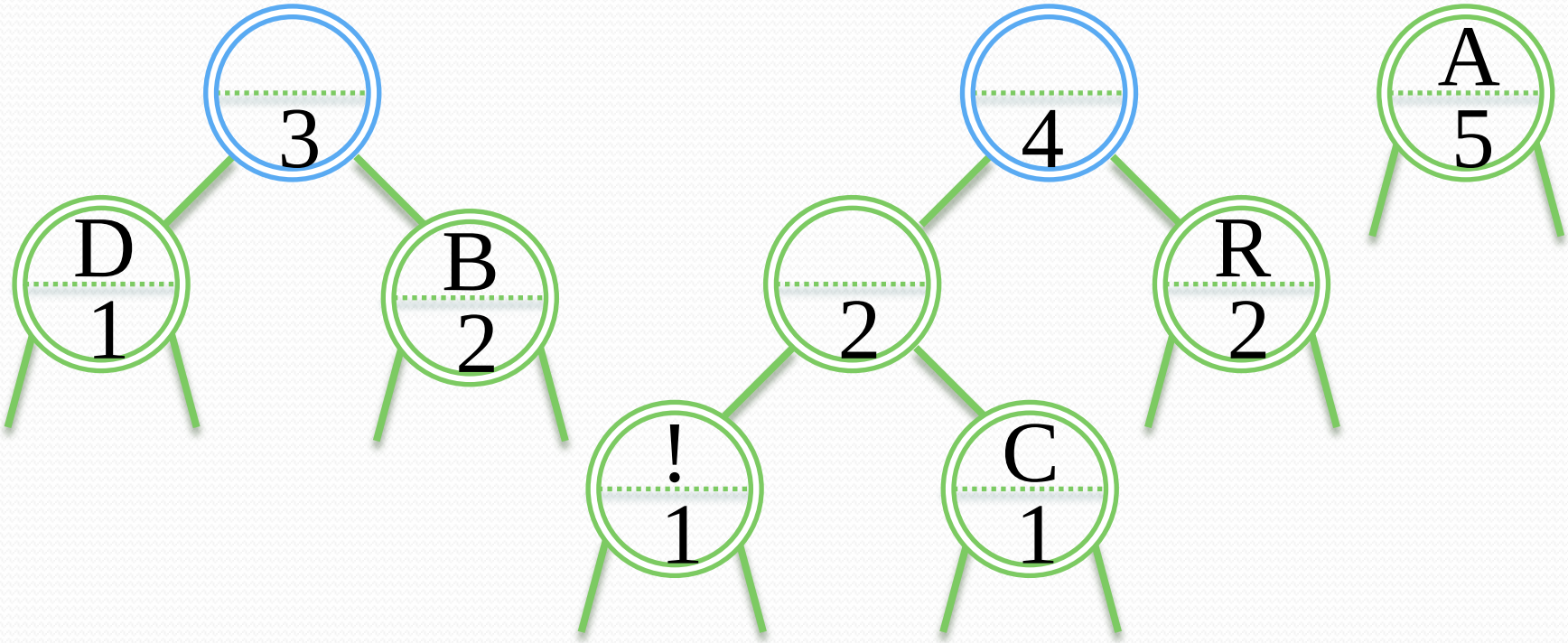
Huffman algorithm: encoding (3/7) Huffman coding tree



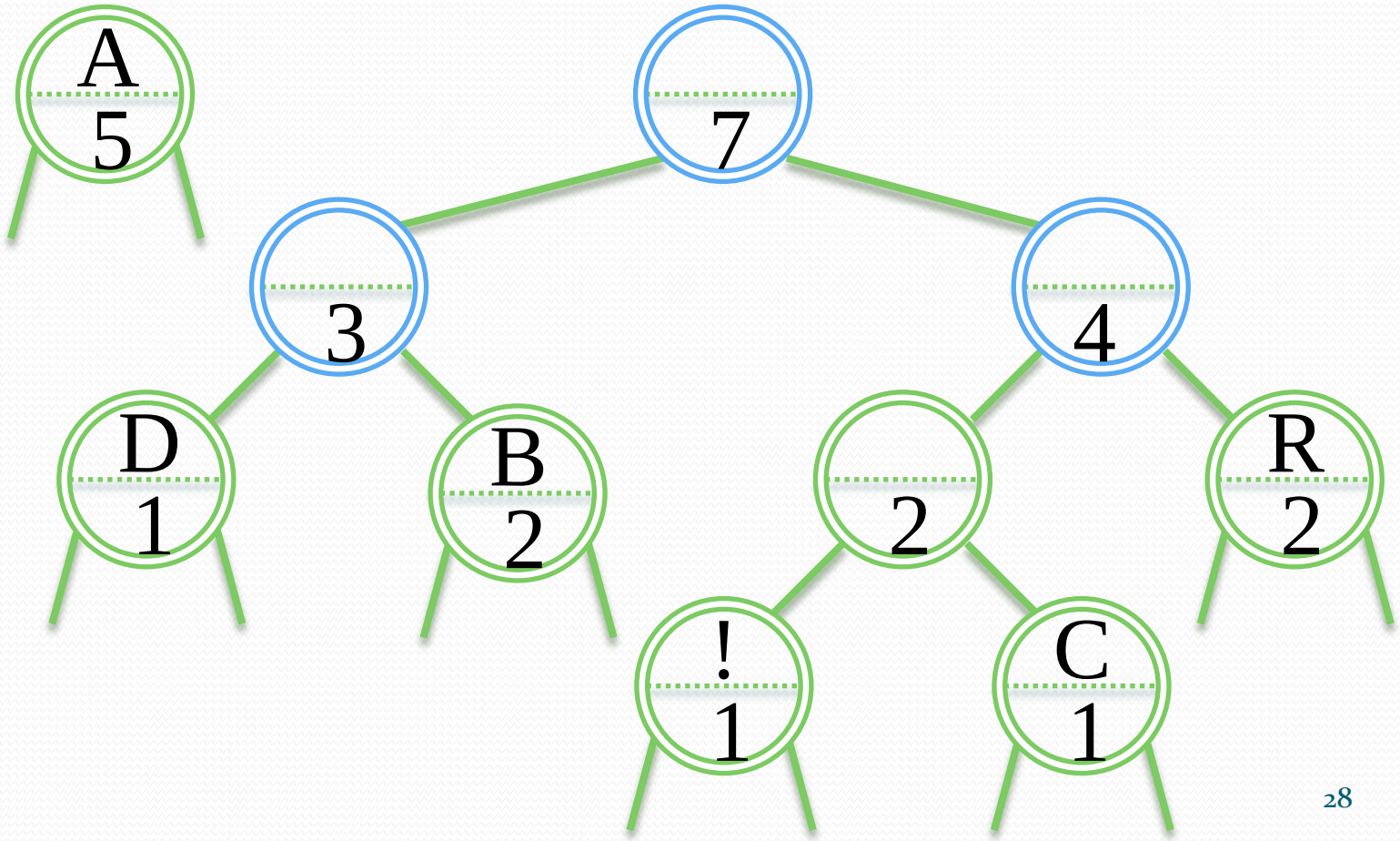
Huffman algorithm: encoding (3/7) Huffman coding tree



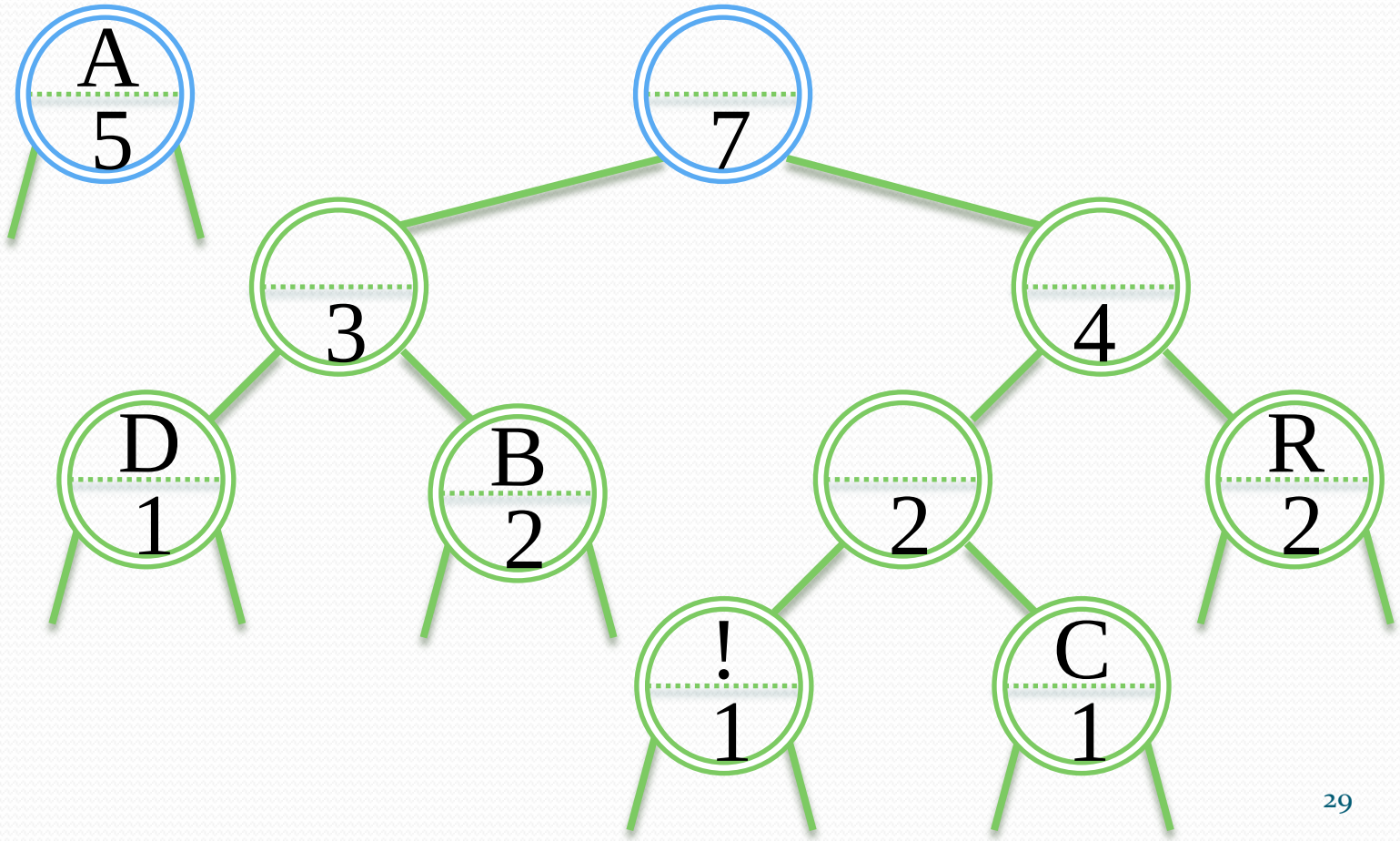
Huffman algorithm: encoding (3/7) Huffman coding tree



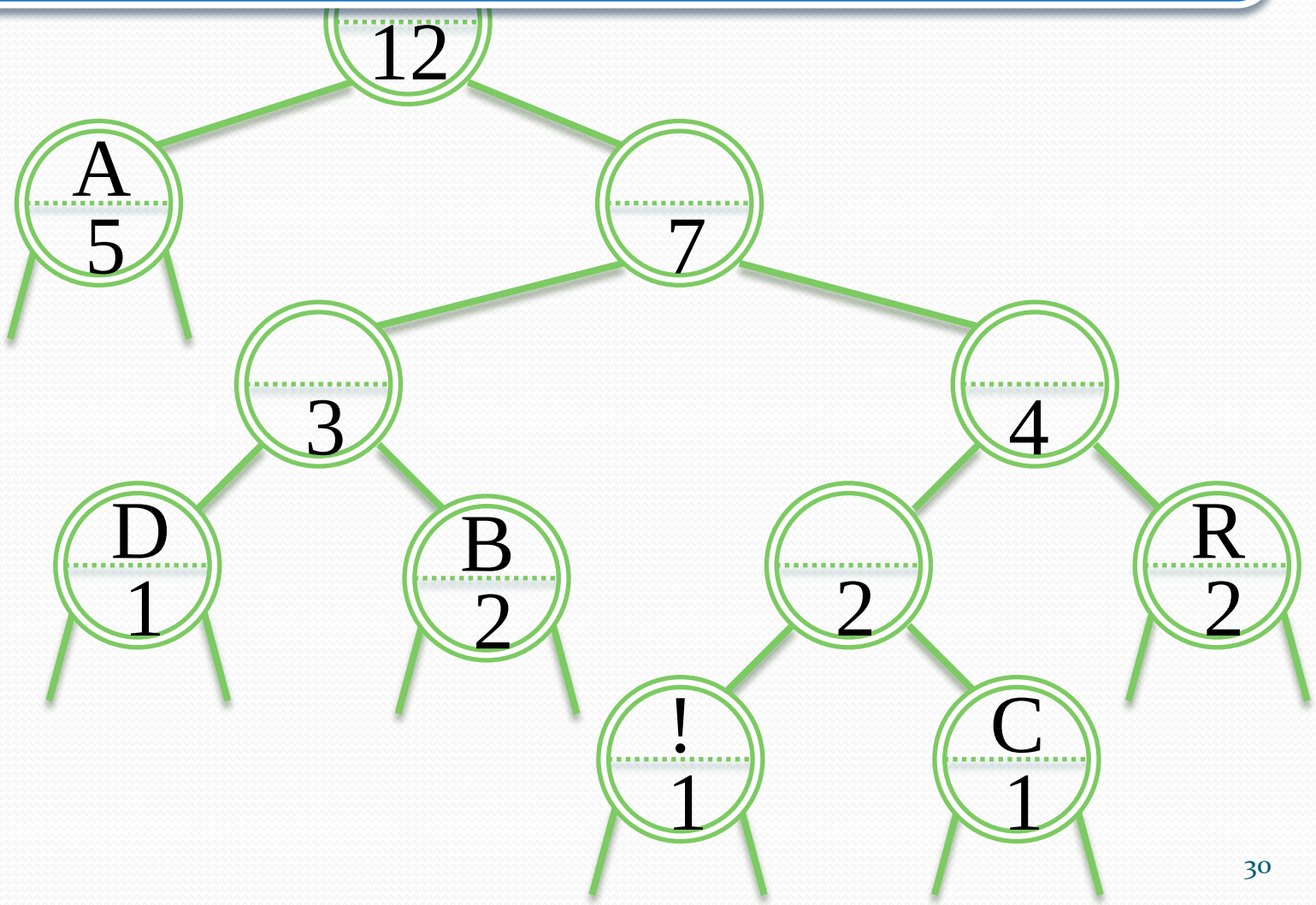
Huffman algorithm: encoding (3/7) Huffman coding tree



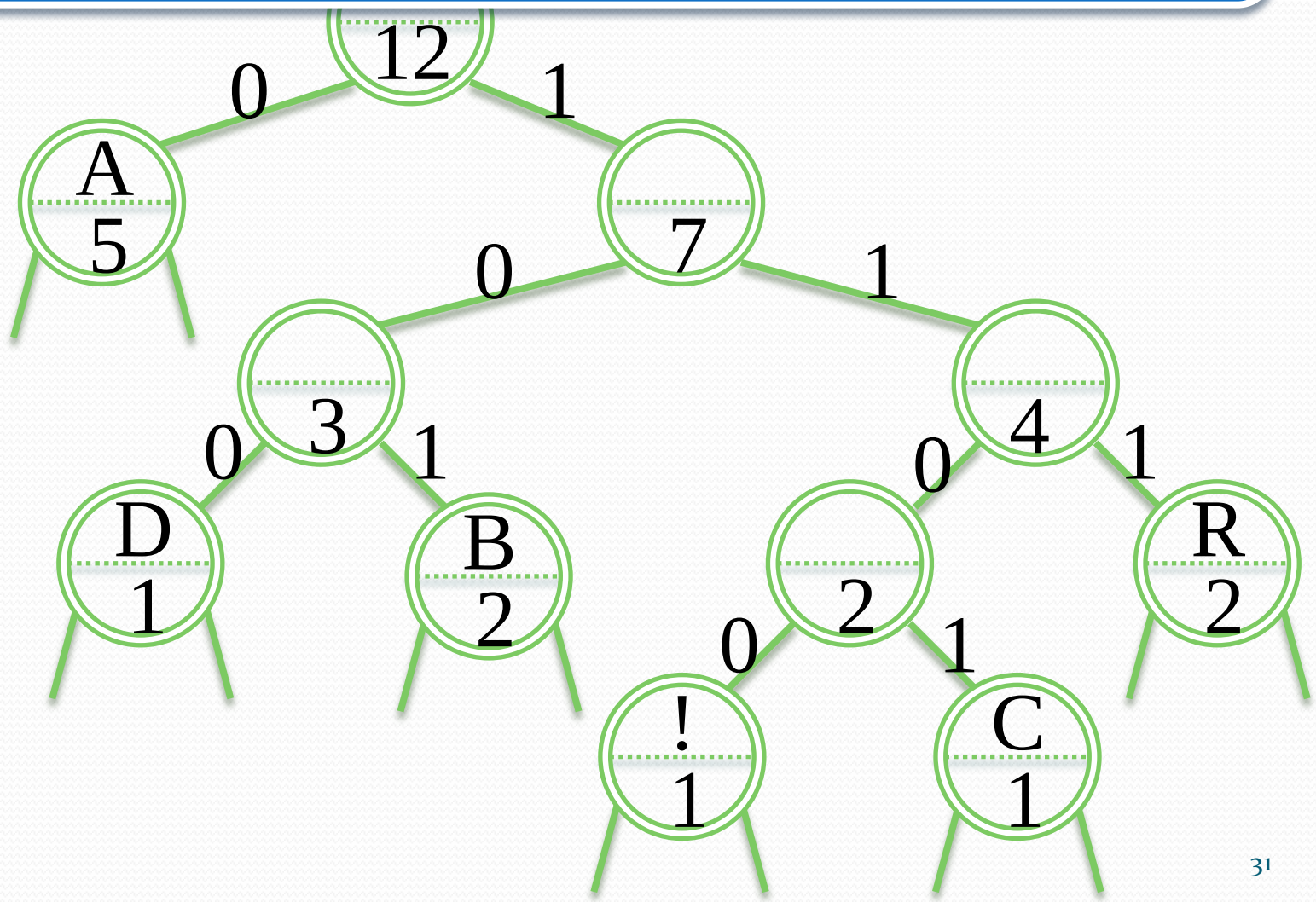
Huffman algorithm: encoding (3/7) Huffman coding tree



Huffman algorithm: encoding (3/7) Huffman coding tree



Huffman algorithm: encoding (4/7) codeword array



Huffman algorithm: encoding (4/7) codeword array

SIGN	CODEWORDS	
	Code page ANSI	PREFIX-FREE CODE
!	00100001	1100
A	01000001	0
B	01000010	101
C	01000011	1101
D	01000100	100
R	01010010	111

Huffman algorithm: encoding
(5/7) record of Huffman's tree (preorder)

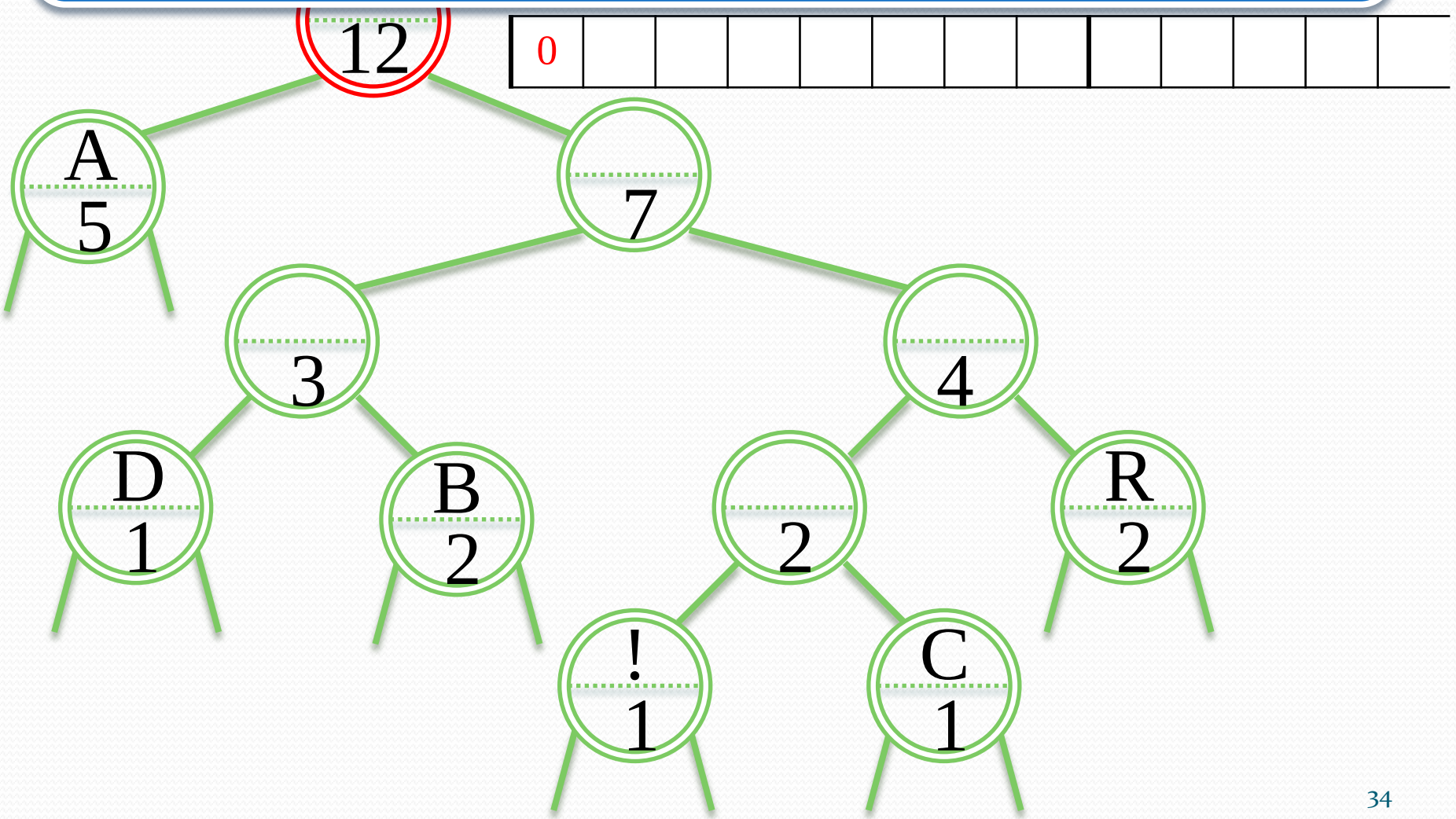
```
void writeTree(Node* x)
{
    if(x->isLeaf())
    {
        ///Zapisz:
        '1'
        'znak'
        return;
    }

    ///Zapisz:
    '0'                 // Wezel wewnetrzny
    writeTree(x->pLeftChild);
    writeTree(x->pRightChild);
}
```

///Write:
'1' // (1 bit it means that it is a leaf)
'sign' // sign stored in the leaf (8 bits
// from code page like „windows-1252")

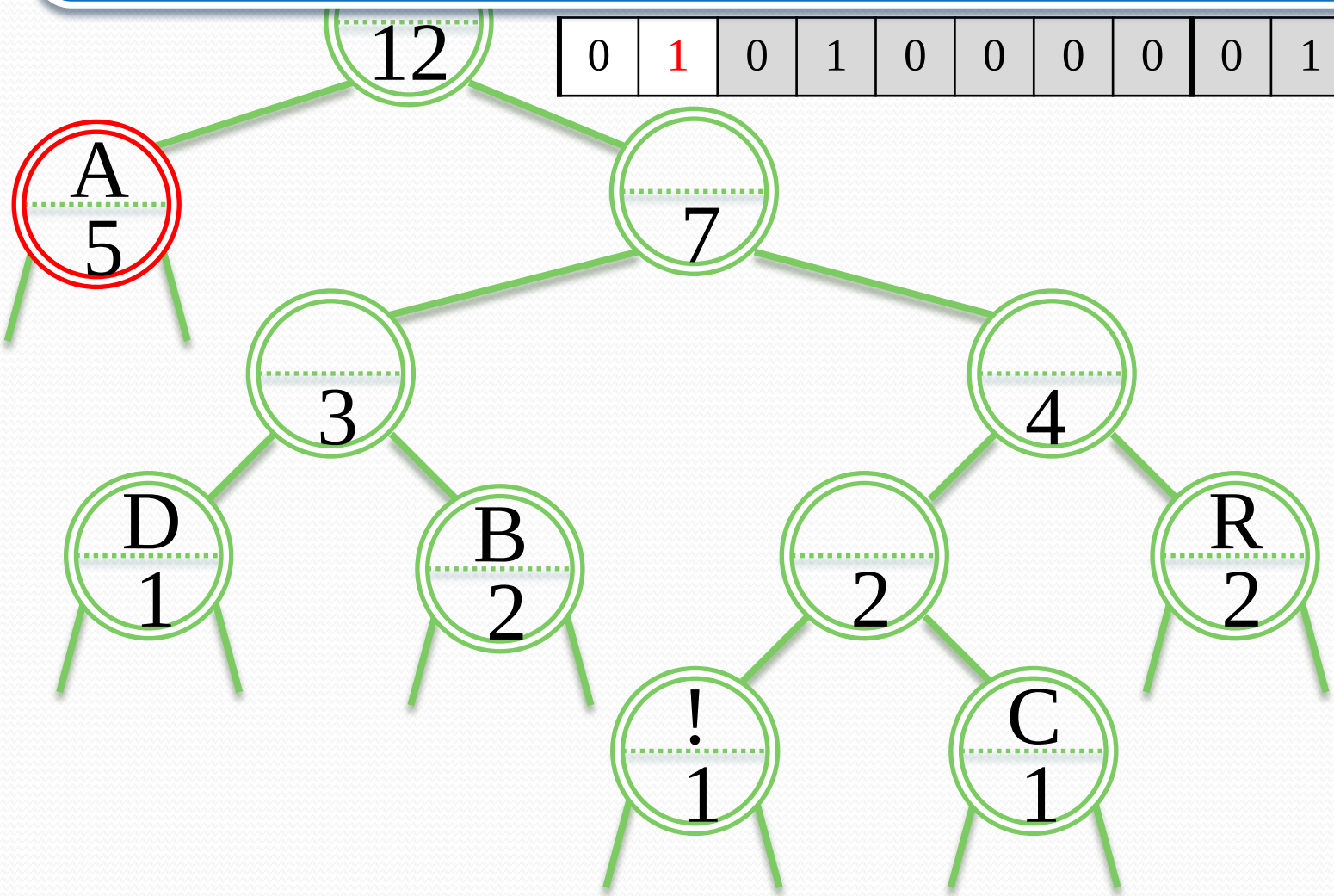
// internal node

Huffman algorithm: encoding

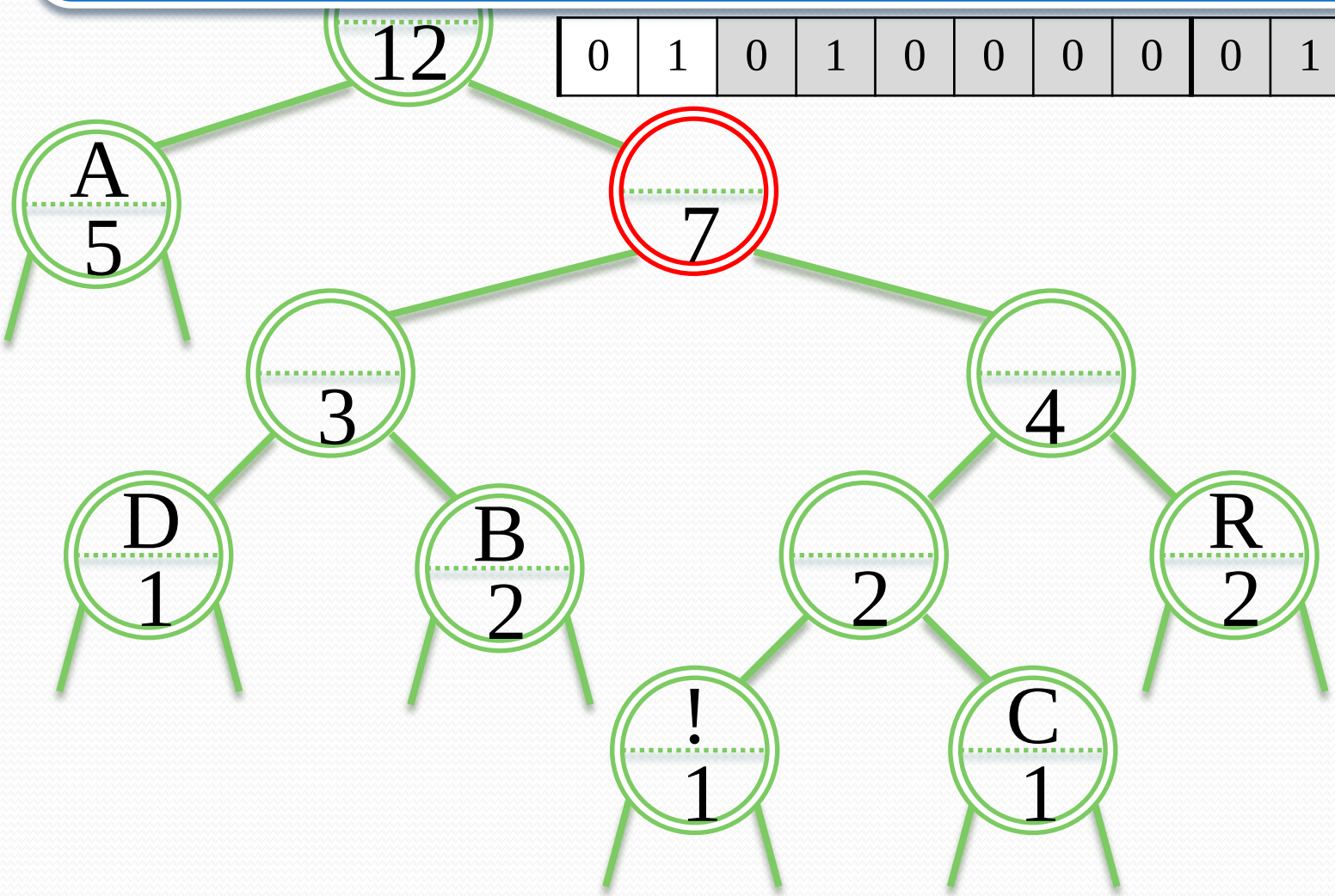
(5/7) record of Huffman's tree (preorder)



Huffman algorithm: encoding
(5/7) record of Huffman's tree (preorder)



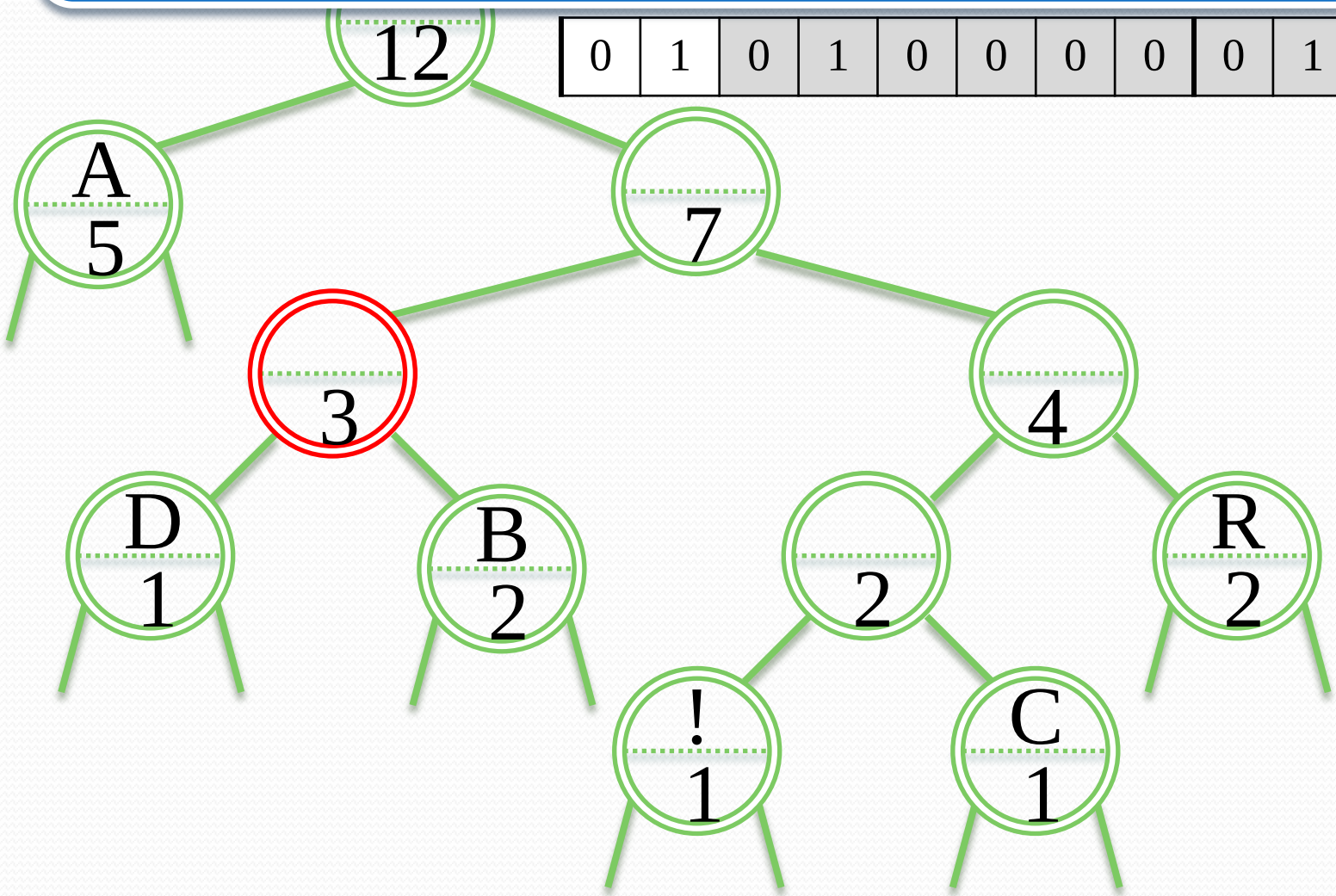
Huffman algorithm: encoding
(5/7) record of Huffman's tree (preorder)



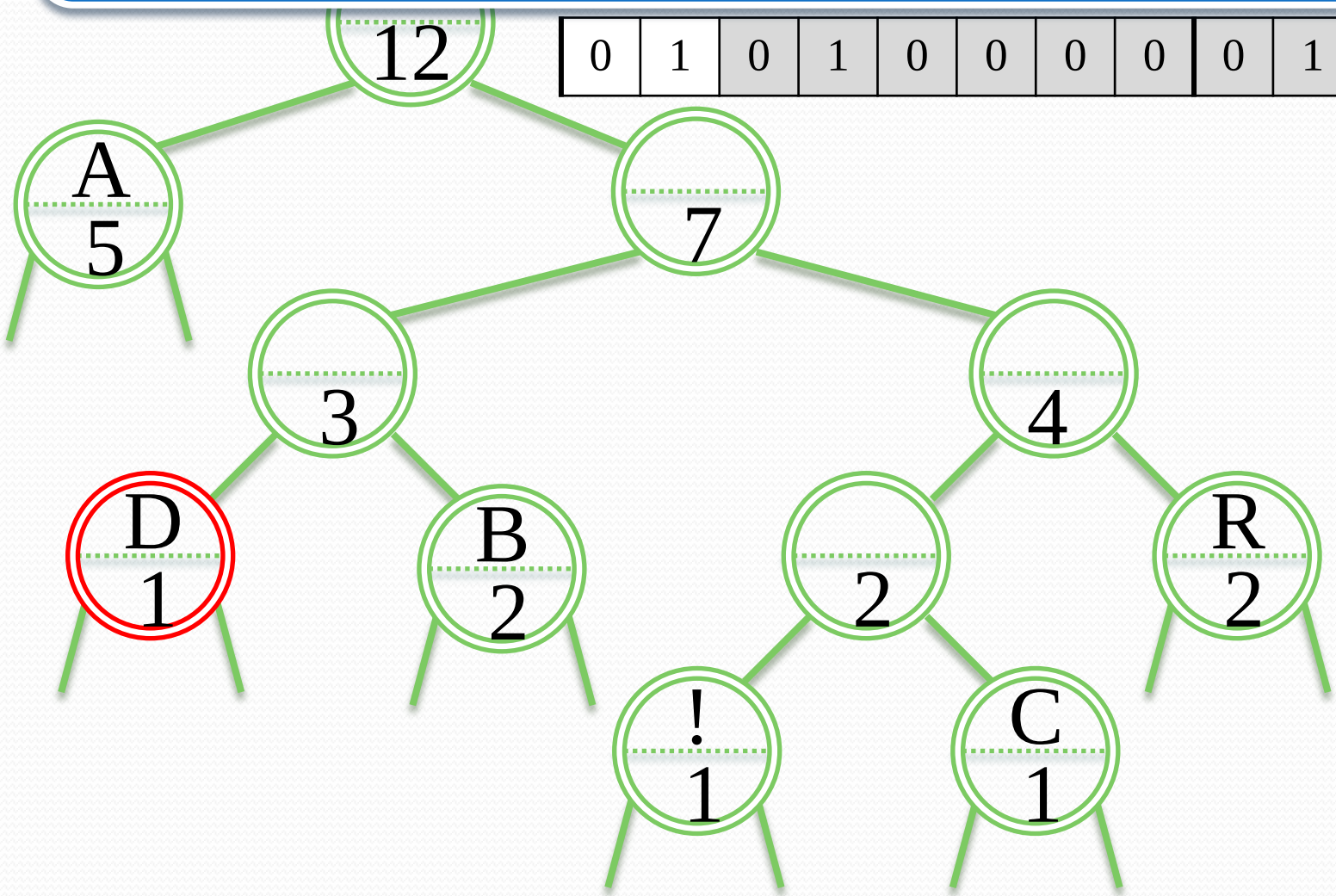
0	1	0	1	0	0	0	0	0	1	0		

Huffman algorithm: encoding

(5/7) record of Huffman's tree (preorder)



Huffman algorithm: encoding
(5/7) record of Huffman's tree (preorder)

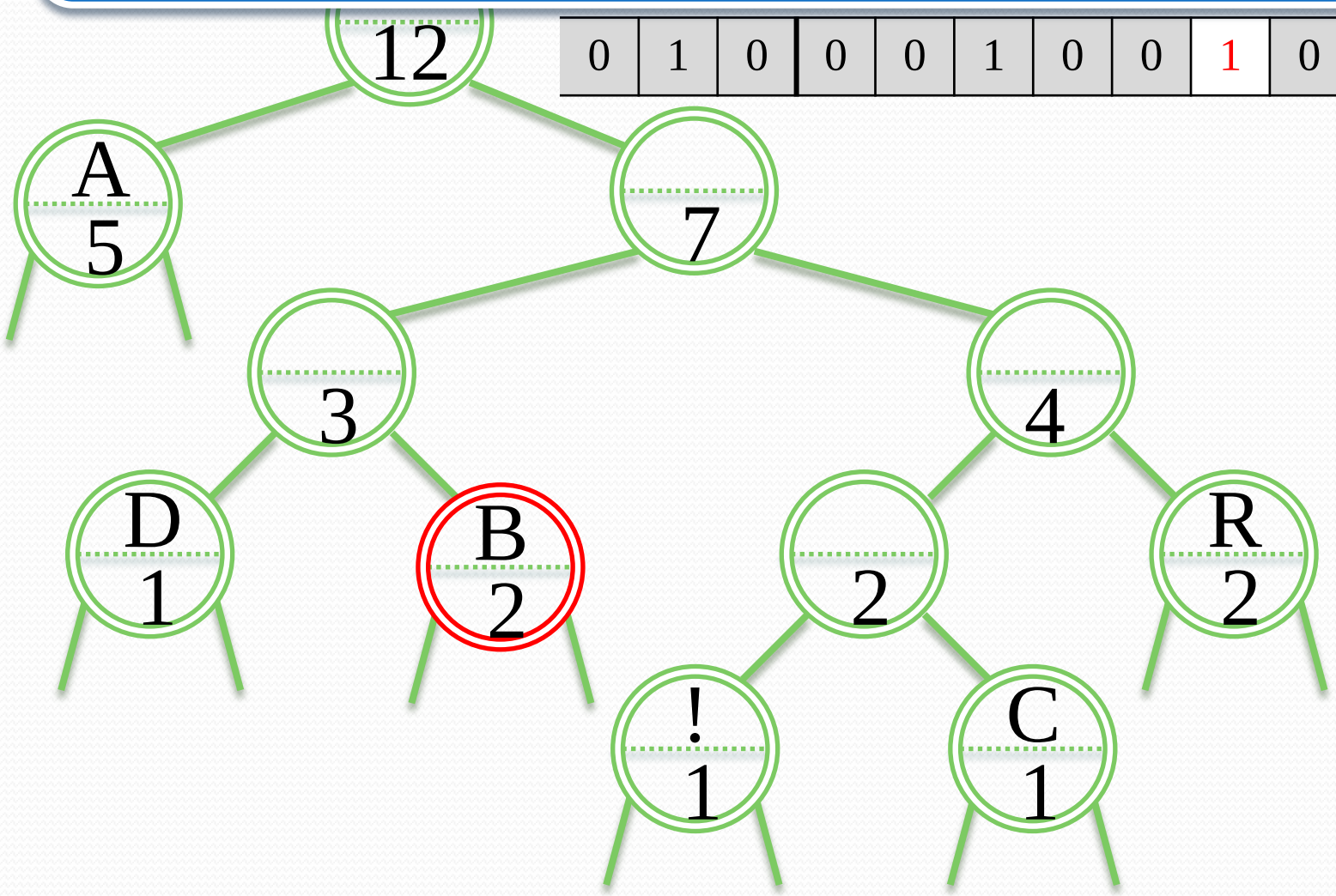


0	1	0	1	0	0	0	0	0	1	0	0	1

Huffman algorithm: encoding

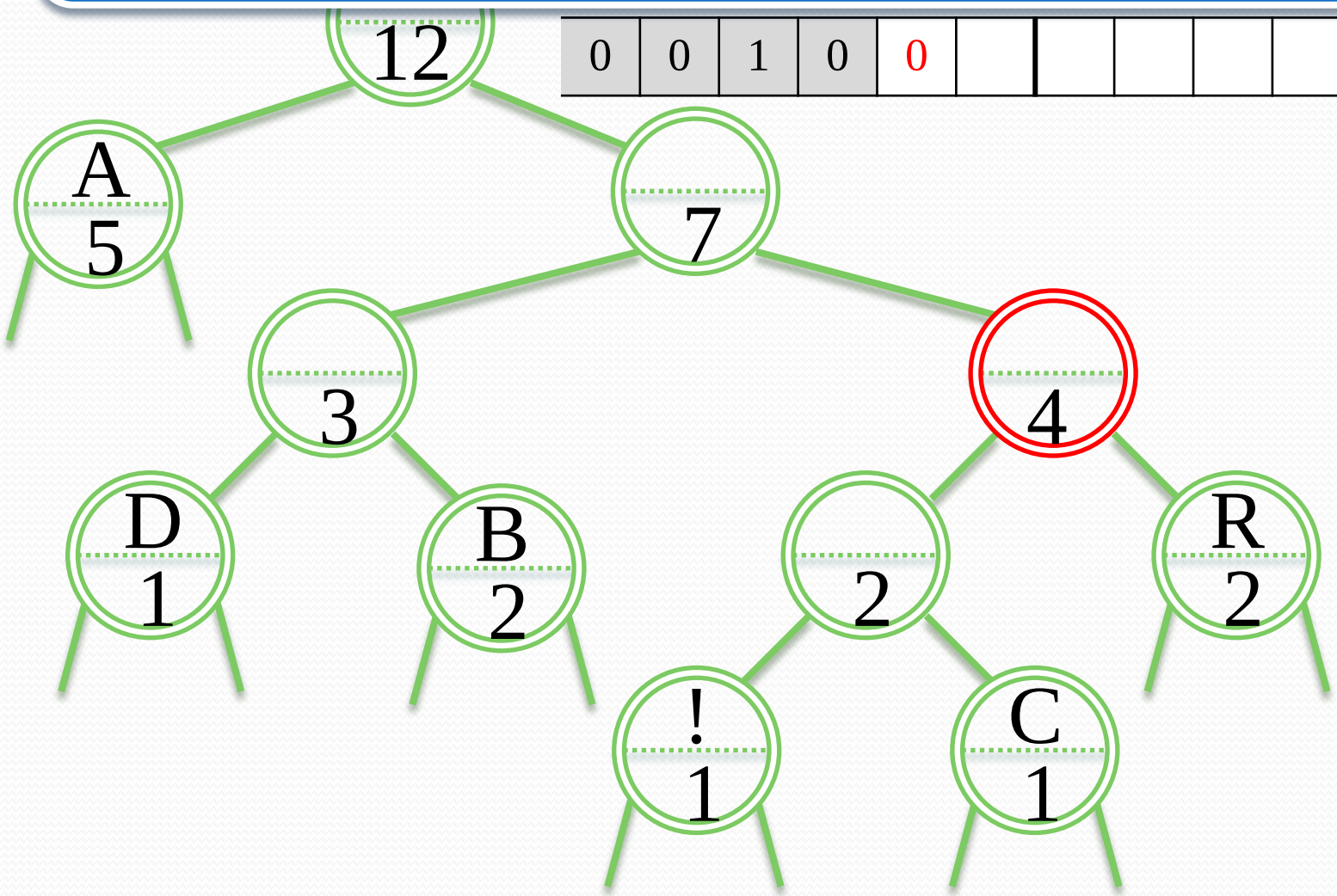
(5/7) record of Huffman's tree (preorder)

0	1	0	0	0	1	0	0	1	0	1	0	0



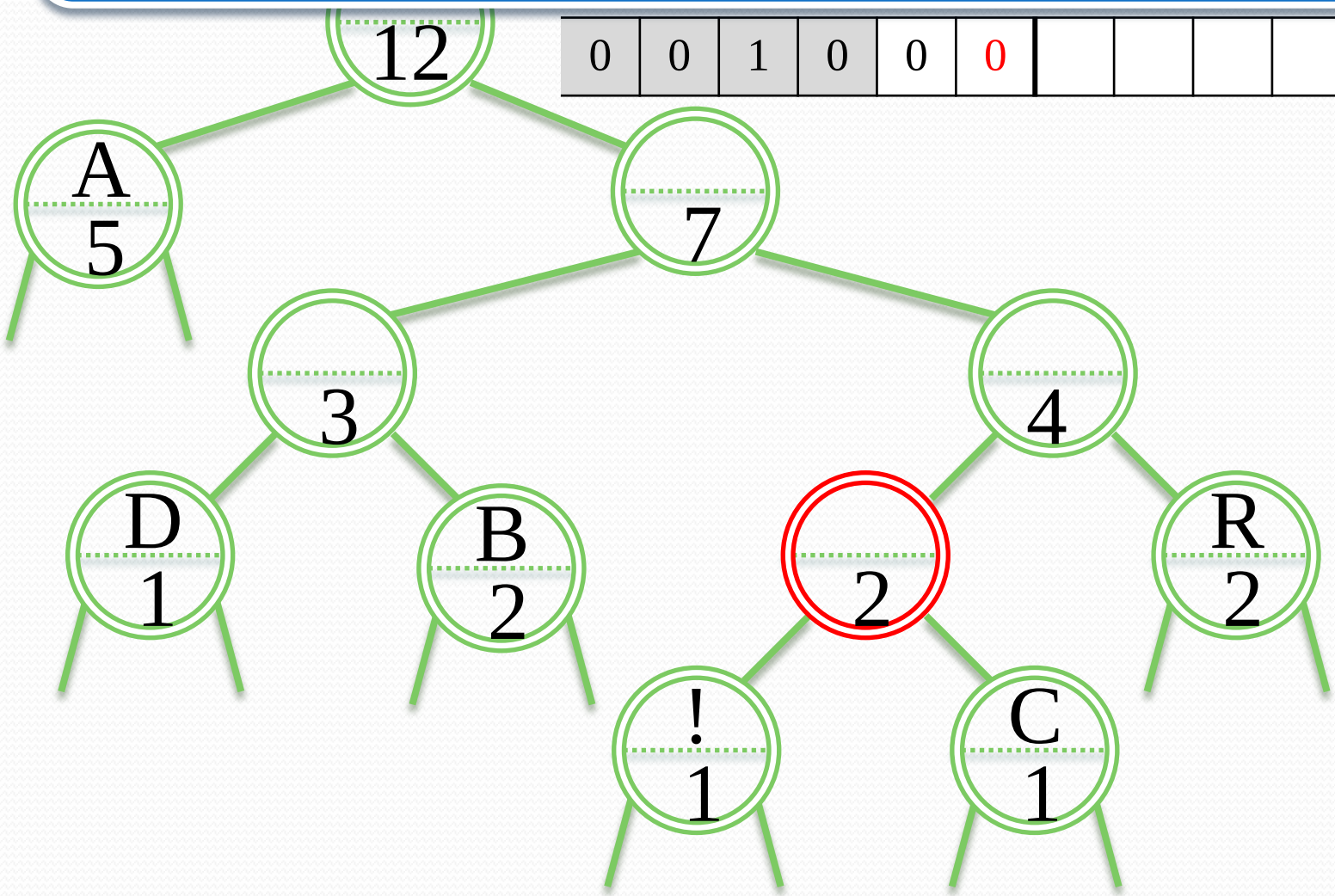
Huffman algorithm: encoding

(5/7) record of Huffman's tree (preorder)



Huffman algorithm: encoding

(5/7) record of Huffman's tree (preorder)

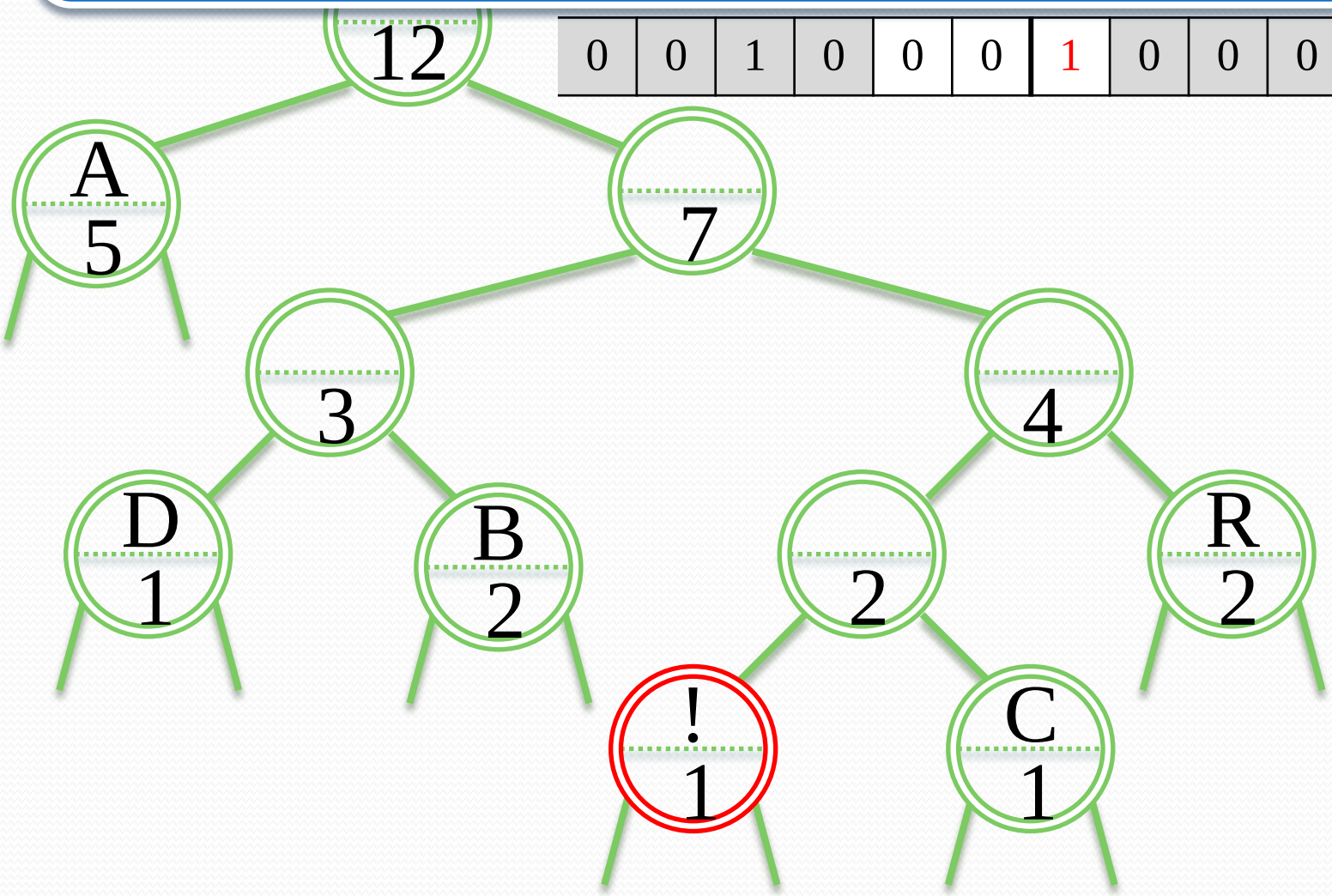


0	0	1	0	0	0							

Huffman algorithm: encoding

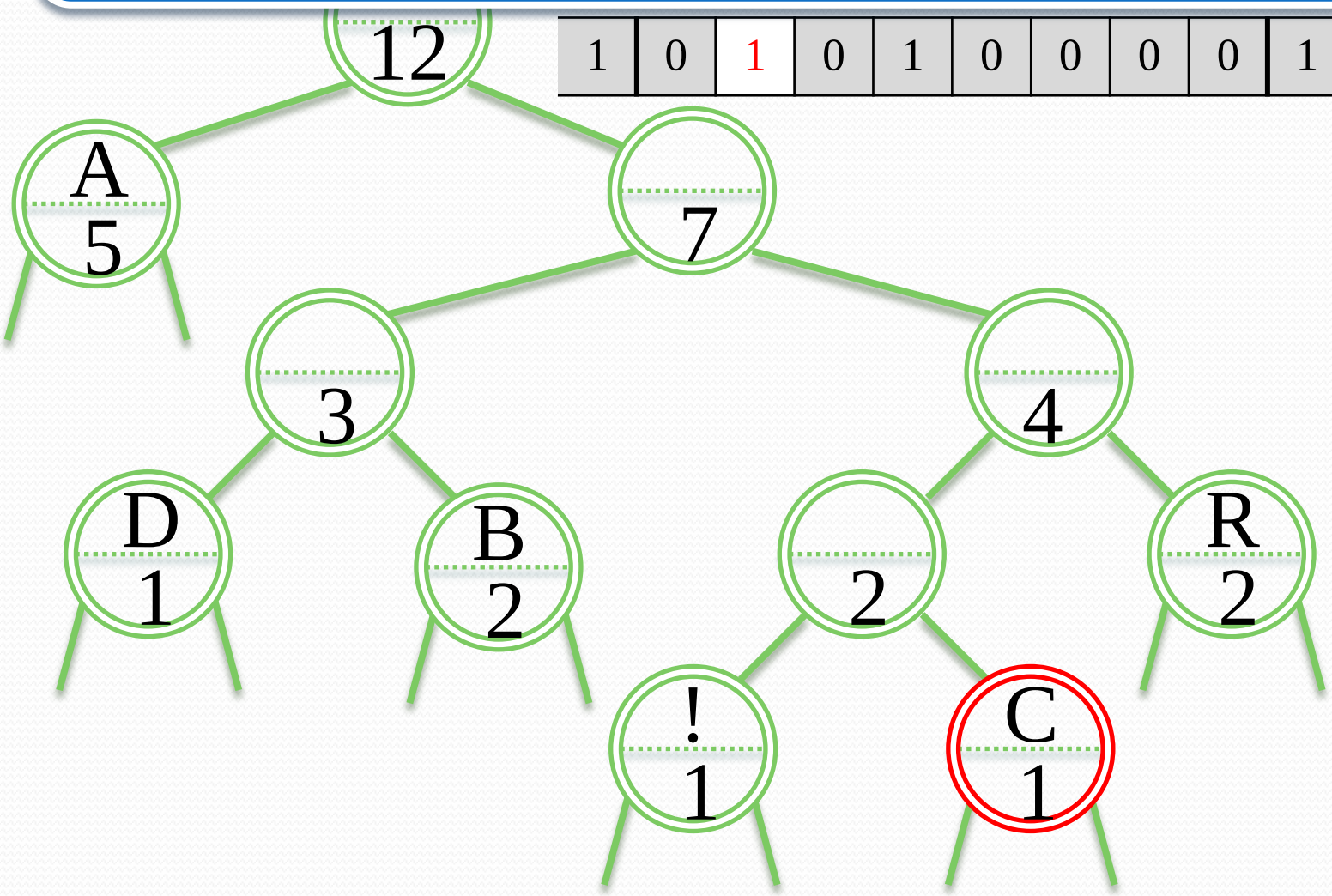
(5/7) record of Huffman's tree (preorder)

0	0	1	0	0	0	1	0	0	0	0	1	0



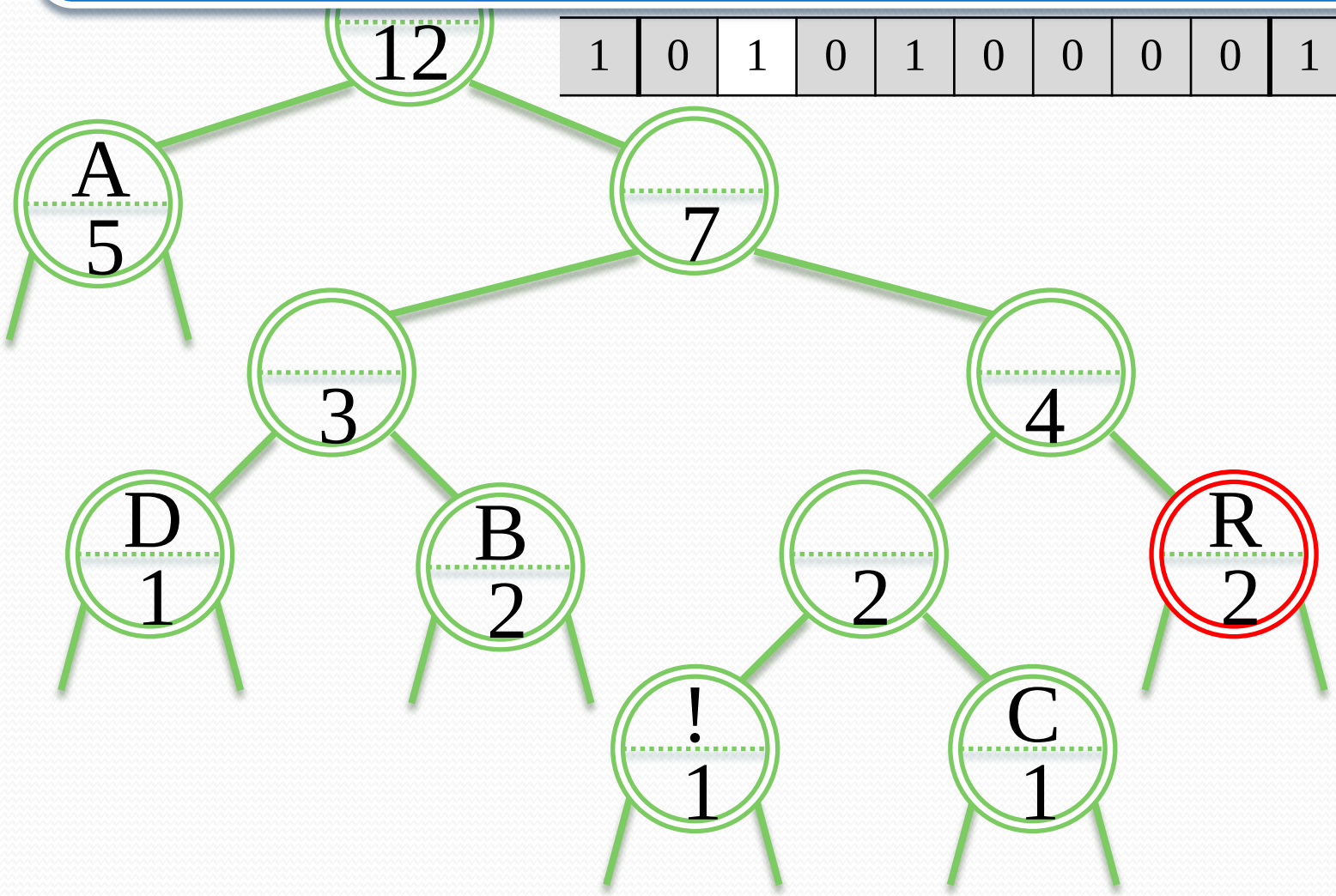
Huffman algorithm: encoding
(5/7) record of Huffman's tree (preorder)

1	0	1	0	1	0	0	0	0	1	1		



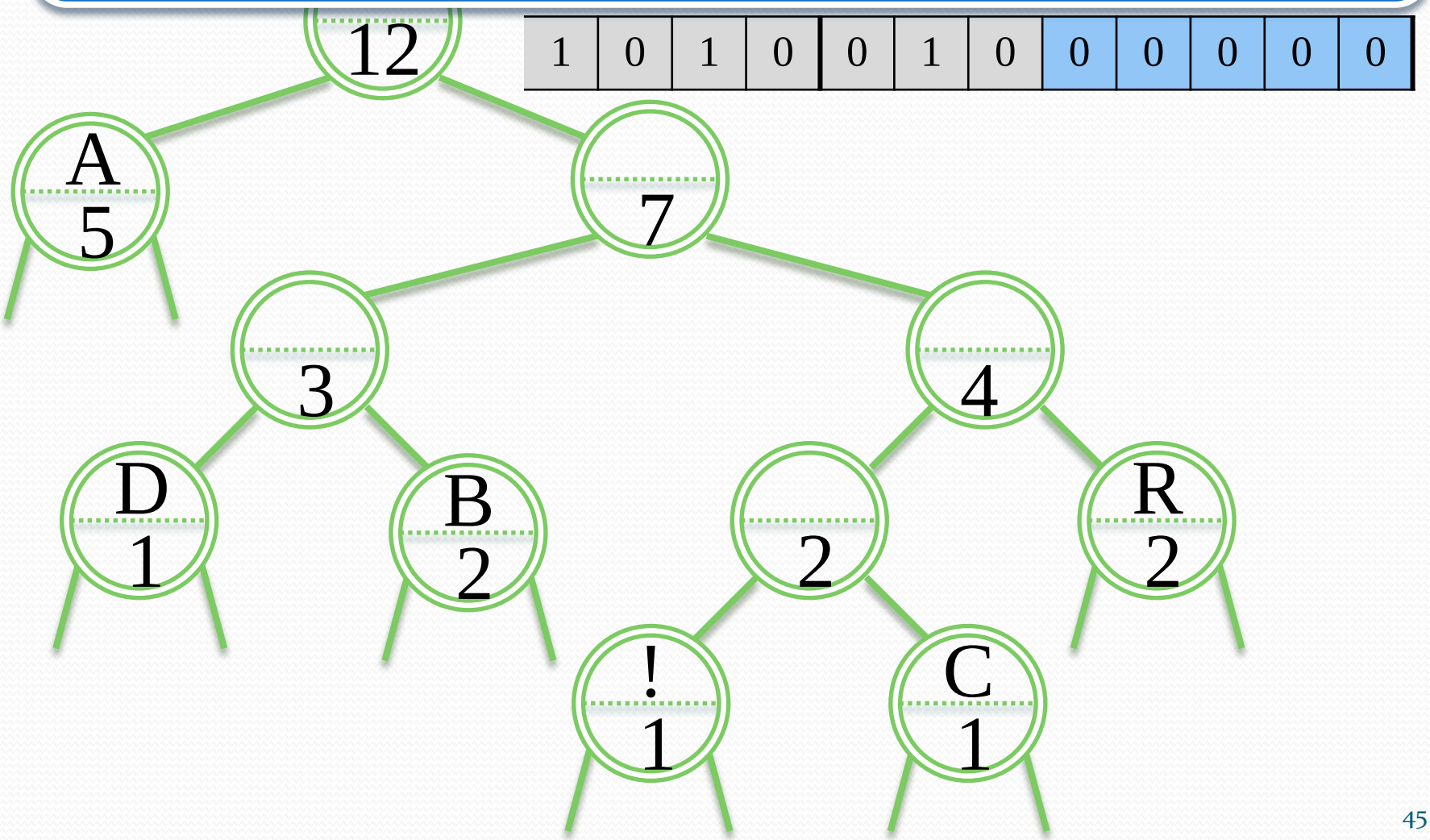
Huffman algorithm: encoding
(5/7) record of Huffman's tree (preorder)

1	0	1	0	1	0	0	0	0	1	1	1	0



Huffman algorithm: encoding

(5/7) record of Huffman's tree (preorder)



Huffman algorithm: encoding

(6/7) recording of the number of characters appearing in the text

1 byte = 2^8 signs = 256 B

2 bytes = 2^16 signs = 64 KB

3 bytes = 2^24 signs = 16 MB

4 bytes = 2^32 signs = 4 GB

5 bytes = 2^40 signs = 1 TB

Huffman algorithm: encoding
(7/7) recording (compression) of input data

SIGN	CODEWORDS		frequency of the sign [f_i]	depth of the character in the tree [i]
	Code page ANSI	PREFIX-FREE CODE		
!	00100001	1100	1	4
A	01000001	0	5	1
B	01000010	101	2	3
C	01000011	1101	1	4
D	01000100	100	1	3
R	01010010	111	2	3

$$Compression\ cost = \sum_{i=1}^{n} f_i * i$$

Huffman algorithm: encoding
(7/7) recording (compression) of input data

File (or sentence)	Compression cost* [byte]	Huffman tree cost* [byte]
„ABRACADABRA!”	4 (28+4 bits)	8 (59+5 bits)
„A TALE OF TWO CITIES”	440702	93
„OPOWIEŚĆ O DWÓCH MIASTACH”	254980	112

* After aligning to full bytes.

Cost of storing the number of characters from the input file: 4 bytes.

Huffman algorithm: encoding
(7/7) recording (compression) of input data

File (or sentence)	File size		compression ratio [%]
	Before encoding [byte]	After encoding [byte]	
„ABRACADABRA!”	12	16	133.33
„A TALE OF TWO CITIES”	773125	440799	57.02
„OPOWIEŚĆ O DWÓCH MIASTACH”	421008	255096	60.59

The less compression ratio we get the less memeory space (compressed file) it get.
But:
The definition is a matter of convention - we could just as well talk about the degree of space saved, then the higher the coefficient, the better the compression algorithm: new_definition [%] = 100% - old_definition_of_compression_ratio.

Huffman algorithm: decoding

1. Read input data.
   a) Read Huffman tree .
   b) Read a number of characters to decode.
2. Use the Huffman tree to decode remained (compressed) data from a file.

Huffman algorithm: decoding
(1a/2) reading the Huffman tree (preorder)

```
Node* readTree()
{
    if(                                    )
    {
        // Znak                                lisciu:
        ch_temp                                 bitow';

        return new Node(ch_temp,0,NULL,NULL);
    }

    return new Node(0,0, readTree(), readTree());
}
```

bit in stream == '1'

// Sign to store in the leaf:
= '8 of consecutive bits';

Huffman algorithm: decoding
(1b/2) reading the number of characters to be decoded

Read 4 bytes.

Huffman algorithm: decoding
(2/2) decoding the bit stream