COMP 550
Algorithm and Analysis

Greedy Algorithms (Huffman Encoding)

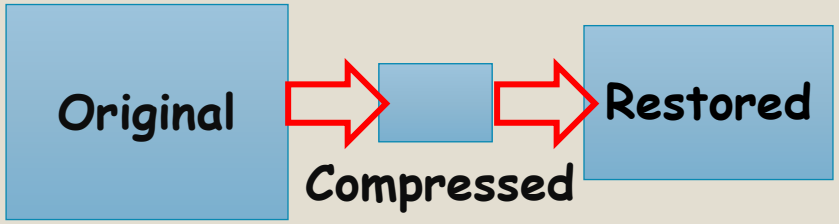
Based on CLRS Sec. 15.3

Some slides are adapted from ones by prior instructors Prof. Plaisted and Prof. Osborne

Data Compression

- Lossy

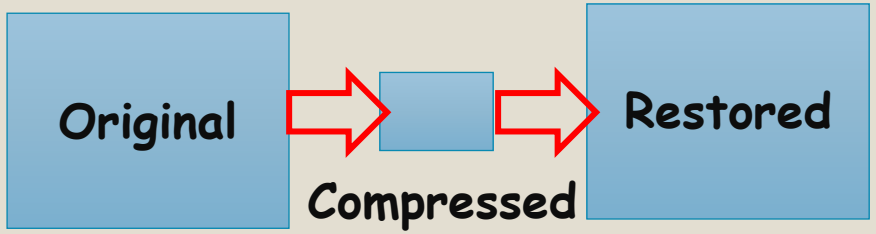
Big file →Small file → Imperfect copy



- Example: JPEG, MP3
- Idea: Targeted data deletion.
  - Delete parts of data that people won't notice.

- Lossless

Big file →Small file → Perfect copy



- Example: ZIP, PNG, JBIG.
- How to do this?

Fixed-Length Encoding

- Let's consider compression of text files
- How is text commonly represented?
  - Fixed length encoding (e.g., ASCII, Unicode)
  - 8 bits in ASCII
- Assuming ASCII, how many bits represent this text (ignore whitespace)?

she sells seashells by the seashore

  - Number of characters: 30
  - Bits required: $30 \times 8 = 240$

Fixed-Length Encoding

- Assuming ASCII, how many bits represent this text (ignore whitespace)?

she sells seashells by the seashore

  - Number of characters: 30
  - Bits required: $30 \times 8 = 240$

- There are 10 different characters in this text
  - At least 4 bits per character are required under fixed-length encoding scheme
  - Bits required: $30 \times 4 = 120$

Variable-Length Encoding

- Drawbacks of fixed-length codes
  - Wasted space
  - Same number of bits used to represent all characters
- **Potential solution**: use variable-length codes
  - Variable number of bits when frequency of occurrence is known
  - Short codes for characters that occur frequently

English letter frequencies (Wikipedia)

Letter	%
E	13
T	9.1
A	8.2
O	7.5
I	7
N	6.7
all others	

Variable-Length Encoding

she sells seashells by the seashore

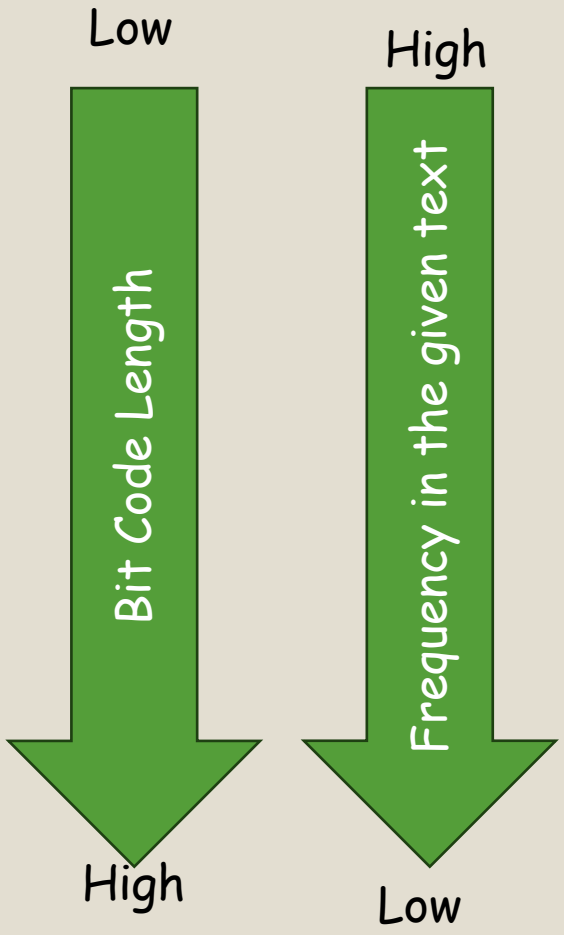
Letter	Freq	Bit code	Bit count	Bits required
E	7	10	2	7x2=14
T	1	00	2	1x2=2
A	2	011	3	2x3=6
O	1	110	3	1x3=3
S	8	0101	4	8x4=32
H	4	0100	4	4x4=16
L	4	11100	5	4x5=20
Y	1	11101	5	1x5=5
B	1	11110	5	1x5=5
R	1	11111	5	1x5=5

Letter	%
E	13
T	9.1
A	8.2
O	7.5
I	7
N	6.7
all others	

Number of bits = 108

Variable-Length Encoding

she sells seashells by the seashore

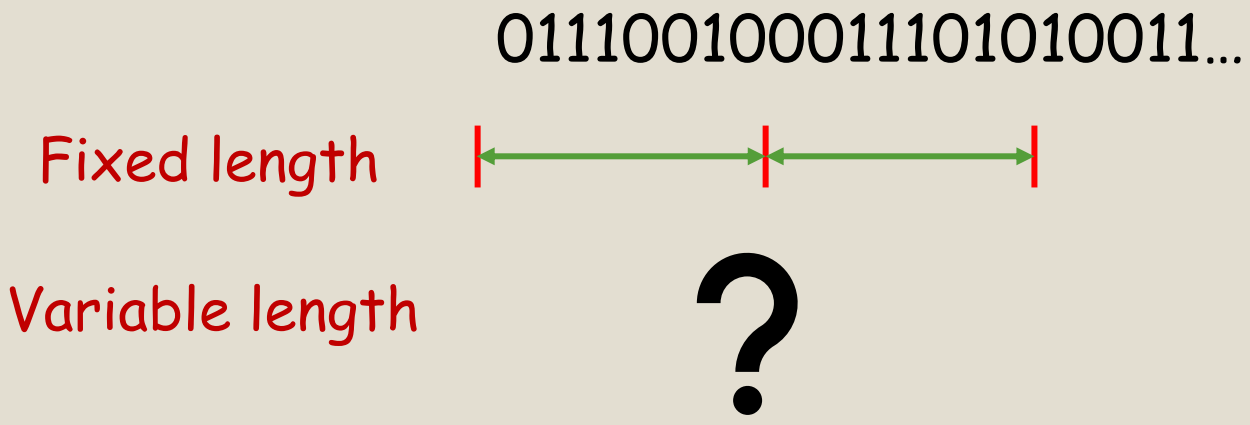


Letter	Freq	Bit code	Bit count	Bits required
S	8	10	2	8x2=16
E	7	00	2	7x2=14
H	4	011	3	4x3=12
L	4	110	3	4x3=12
A	2	0101	4	2x4=8
R	1	0100	4	1x4=4
B	1	11100	5	1x5=5
Y	1	11101	5	1x5=5
T	1	11110	5	1x5=5
O	1	11111	5	1x5=5

Number of bits = 86

Variable-Length Encoding

- **One issue:** Where a character ends and another begins?
  - Not a problem under fixed-length encoding



Prefix-Free Codes

- No codeword is a prefix of another codeword

No other codeword starts with 10

Letter	Freq	Bit code	Bit count	Bits required
S	8	10	2	8x2=16
E	7	00	2	7x2=14
H	4	011	3	4x3=12
L	4	110	3	4x3=12
A	2	0101	4	2x4=8
R	1	0100	4	1x4=4
B	1	11100	5	1x5=5
Y	1	11101	5	1x5=5
T	1	11110	5	1x5=5
O	1	11111	5	1x5=5

Code: 0110011011011111

HELLO

Prefix-Free Codes

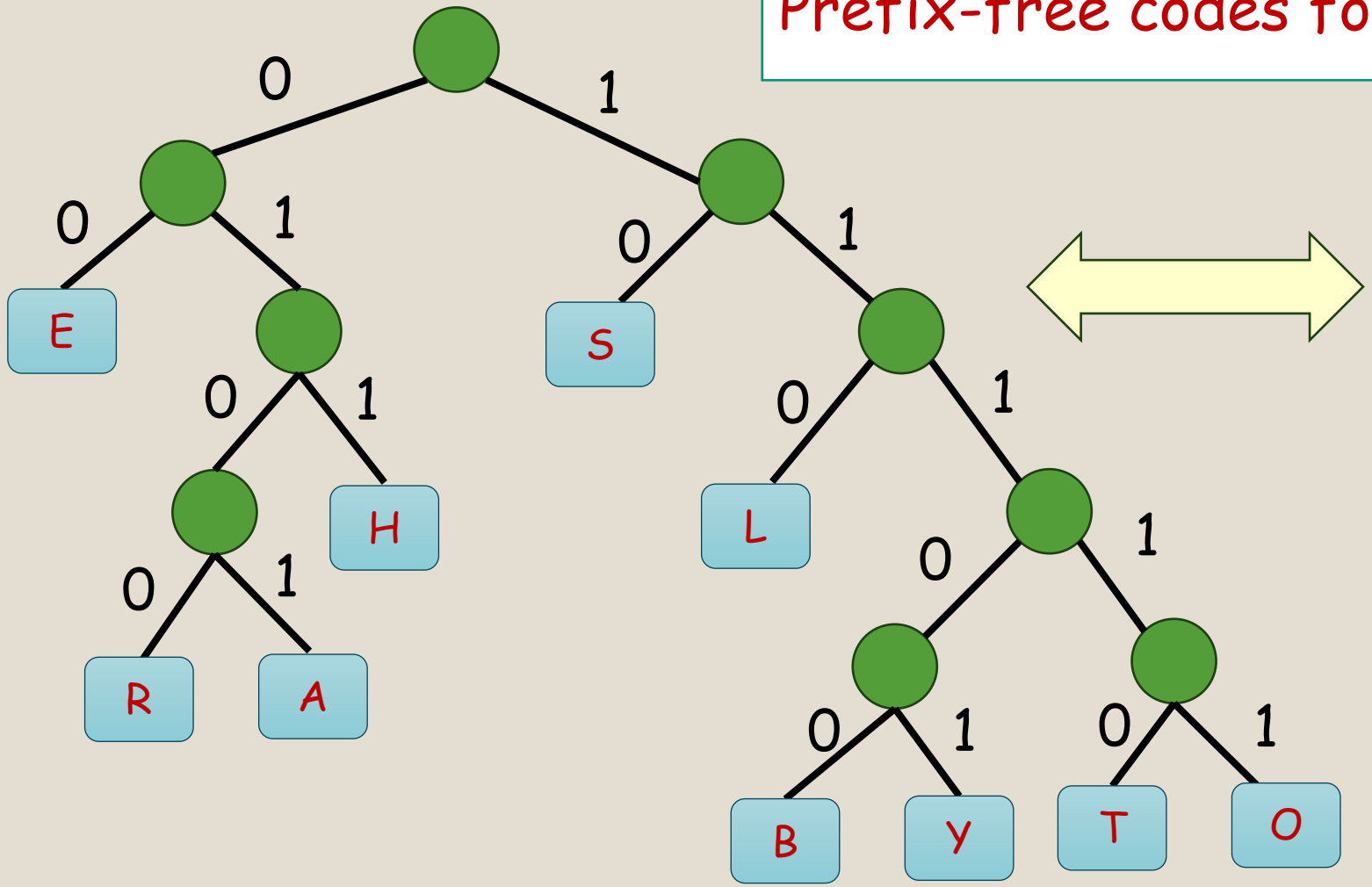
- The following codes do not satisfy prefix-free property

Letter	Bit code
S	0
E	1
H	01
L	10

- Decode the code 1010
  - Can be anyone among "ESES, LL, LES, ESL"

Prefix-Free Codes

Prefix-free codes form a binary tree



Letter	Freq	Bit code
S	8	10
E	7	00
H	4	011
L	4	110
A	2	0101
R	1	0100
B	1	11100
Y	1	11101
T	1	11110
O	1	11111

Optimal Encoding

- Problem: Determine optimal prefix-free codes for a given text
  - Optimal = Minimize the number of bits to store the text
  - Minimize: $\sum$ frequecy of a char $\times$ Number of bits for the char
- Key property:
  - Less frequent characters are further down the tree

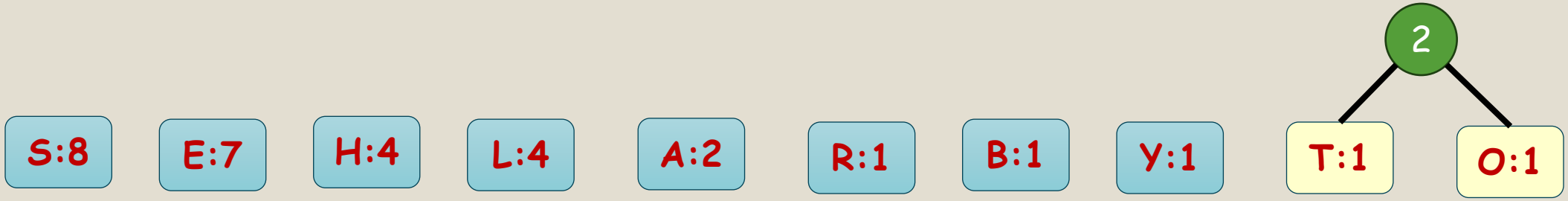
Huffman Encoding

1. Tabulate character frequencies

2. Each **char:frequency** tuple forms a single-node tree

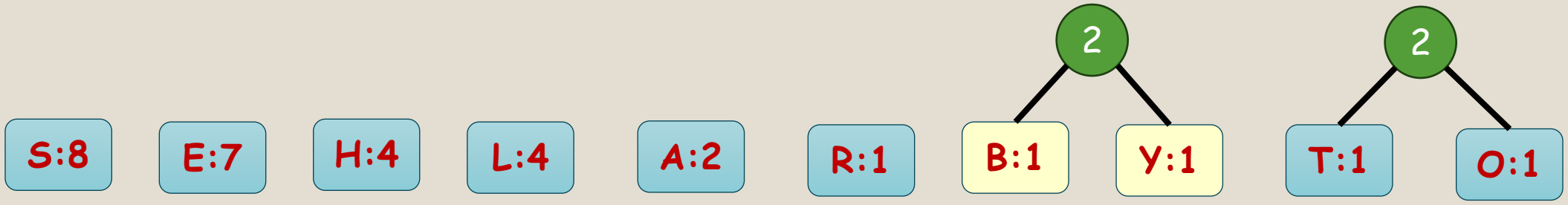
S:8 E:7 H:4 L:4 A:2 R:1 B:1 Y:1 T:1 O:1

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



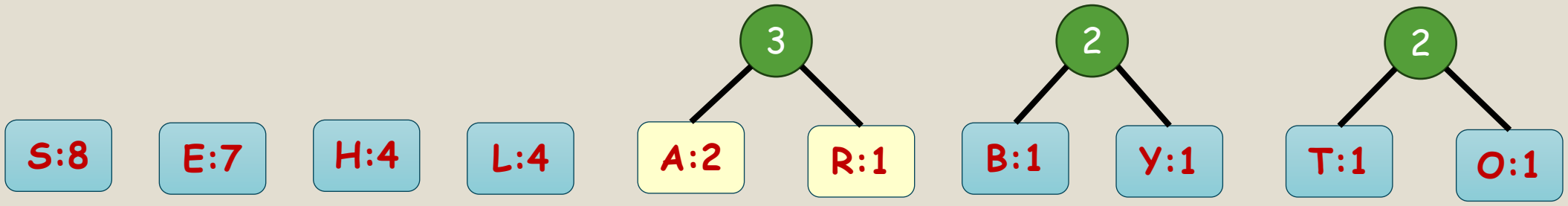
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



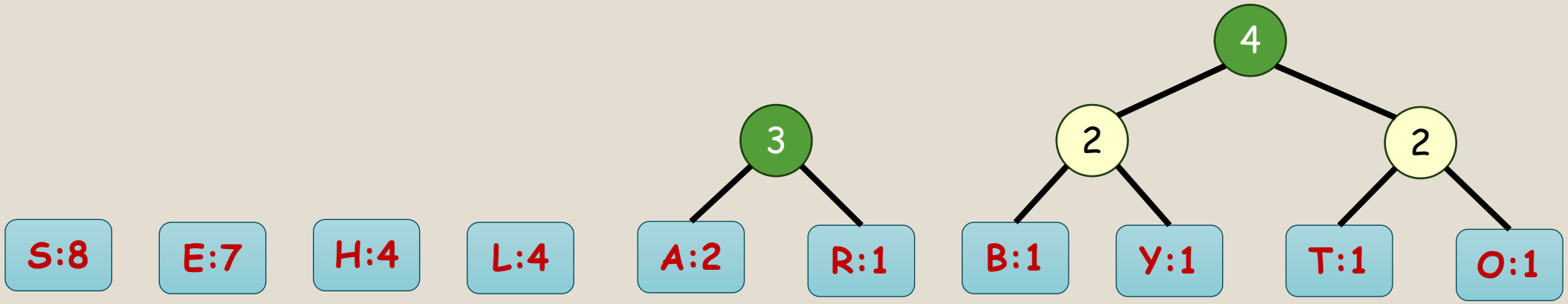
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



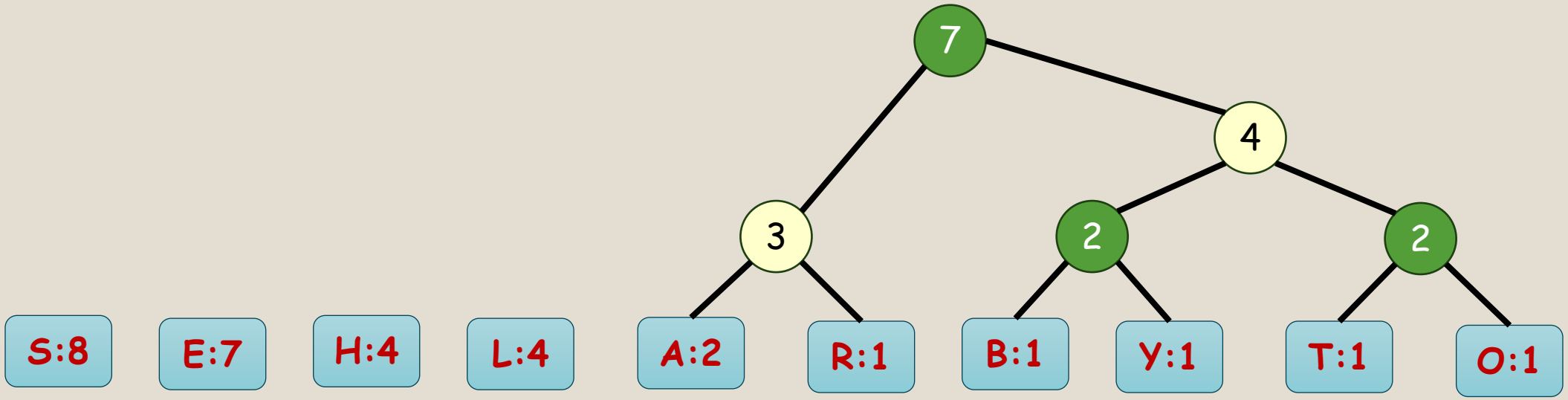
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



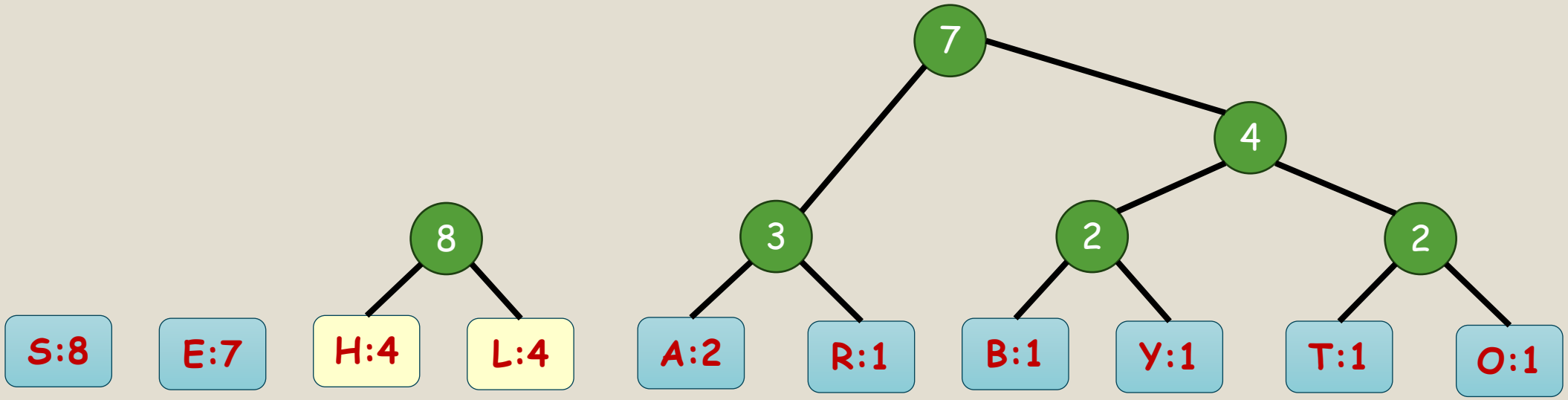
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



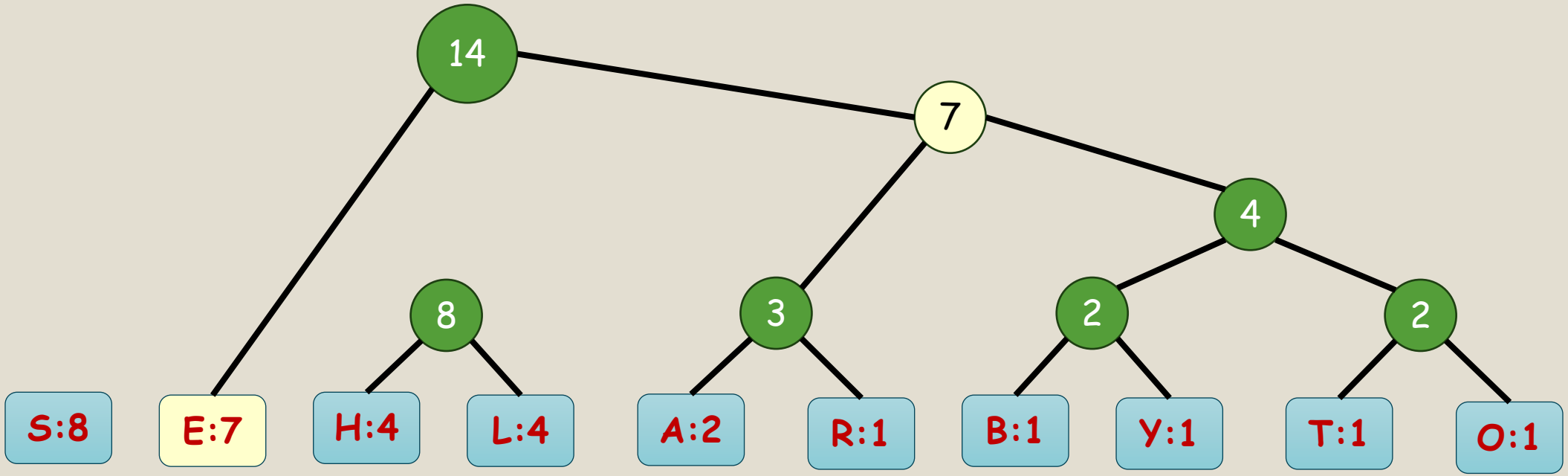
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



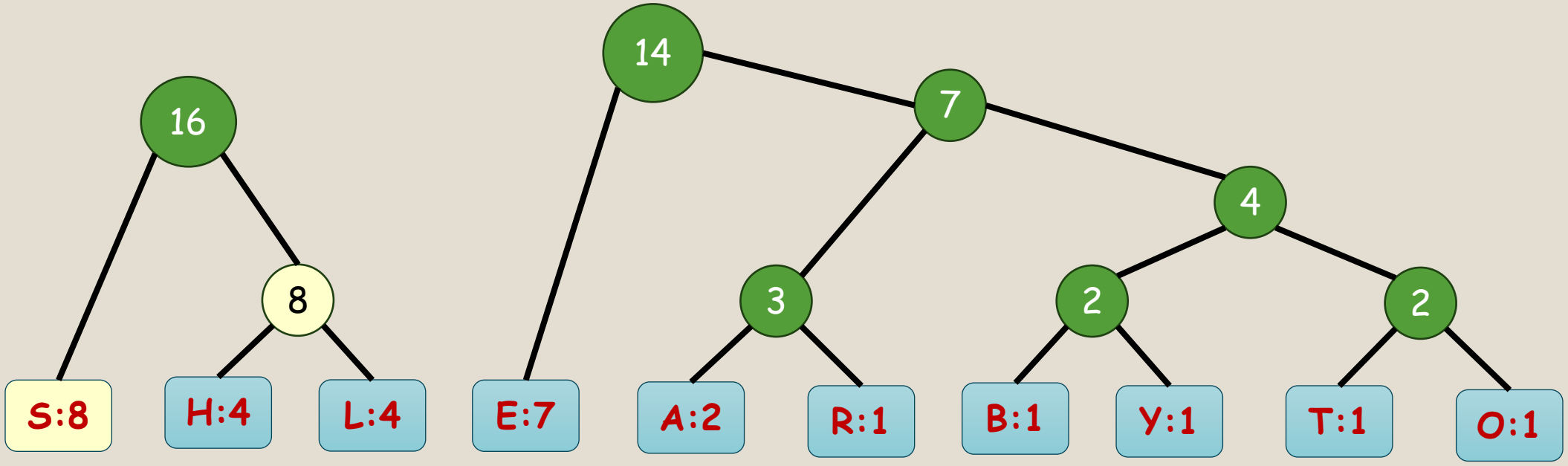
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



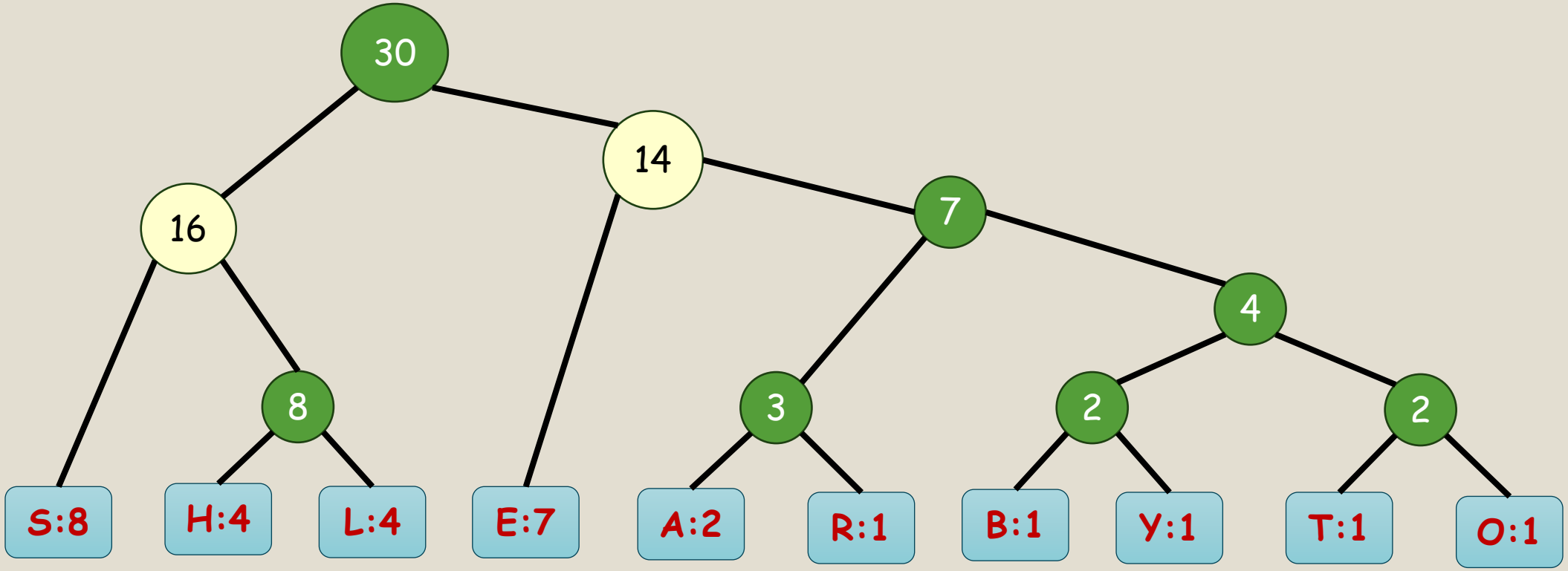
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



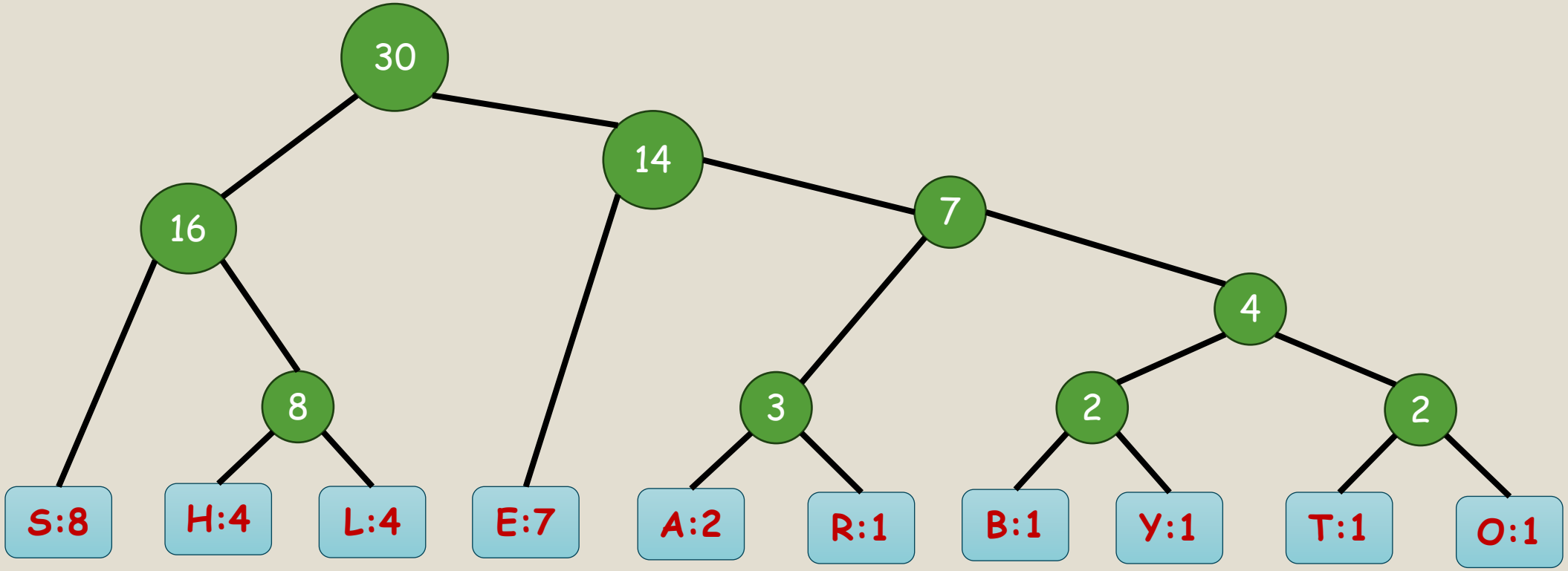
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



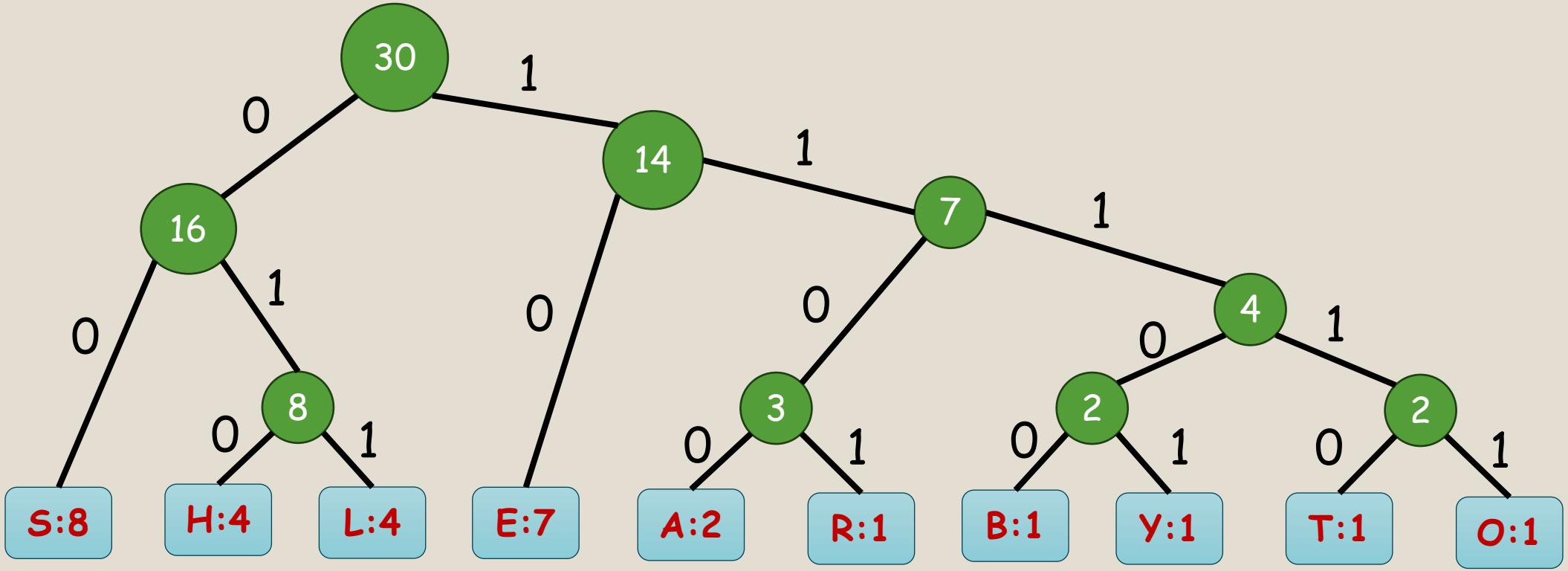
Huffman Encoding

3. Merge the two trees with the smallest roots. The new root is the sum of the two frequencies below it. Repeat until there's one tree.



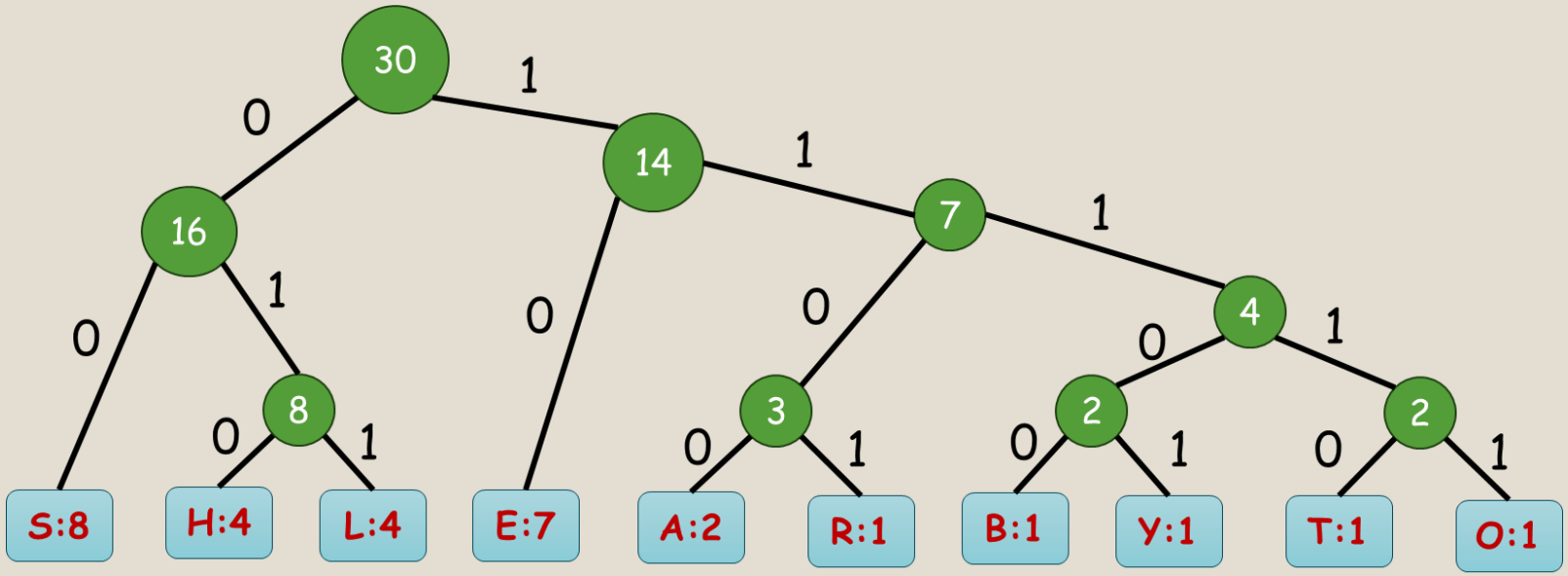
Huffman Encoding

4. Label left edges "0" and right edges "1"



Huffman Encoding

Codes are given by the path from root to leaf



Letter	Bit code
S	00
E	10
H	010
L	011
A	1100
R	1101
B	11100
Y	11101
T	11110
O	11111

Huffman Encoding

- Input: A set C of n characters each with an attribute $freq$
- Output: A tree corresponding to optimal code

```
HUFFMAN(C)
 1 n = |C|
 2 Q = C
 3 for i = 1 to n − 1
 4     allocate a new node z
 5     x = EXTRACT-MIN(Q)
 6     y = EXTRACT-MIN(Q)
 7     z.left = x
 8     z.right = y
 9     z.freq = x.freq + y.freq
10     INSERT(Q, z)
11 return EXTRACT-MIN(Q)  // the root of the tree is the only node left
```

$O(\lg n)$ when we use MIN-Priority Queue

Running time $O(n \lg n)$

Correctness

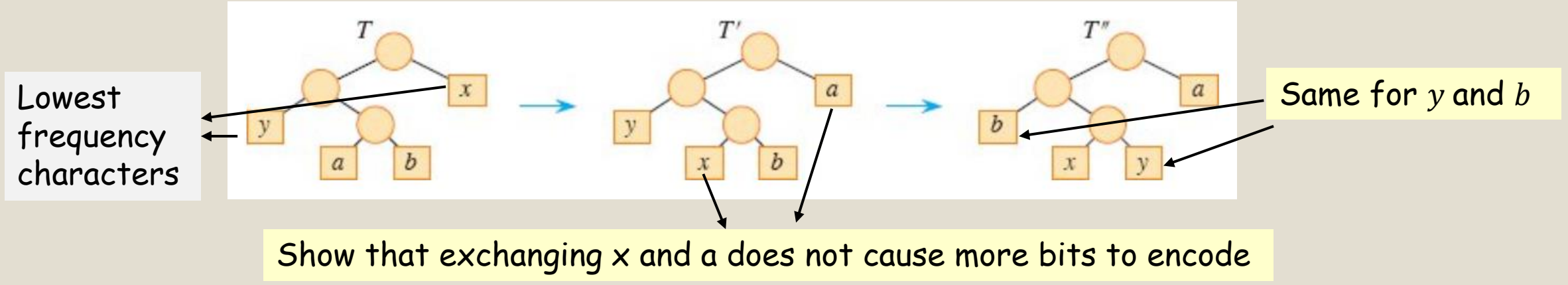
- To show that Huffman encoding builds an optimal prefix-free code:
  - The greedy choice of merging two trees with the smallest root satisfies the greedy-choice property
  - Constructing optimal prefix-free codes has the optimal substructure property

Greedy-Choice Property

There is an optimal solution containing the greedy choice (lowest frequency characters are siblings at the maximum depth)

Proof (sketch) by exchange argument:

- Take a tree T representing an arbitrary optimal prefix-free code
- Transform the tree T to a tree T' that has the greedy choice made by Huffman algorithm
- Show that T' is no worse than T



Optimal Substructure Property

Making the least frequency characters siblings and solving the new subproblem results in an optimal prefix-free code.

Goal: Replace the least frequency character x and y and a new node z with

$$z.freq = x.freq + y.freq$$

Optimal solution of problem = optimal solution of subproblem + (x and y as children of z)

Proof sketch: Assume that

Optimal solution of problem $\neq$ optimal solution of subproblem + (x and y as children of z)

- By the greedy-choice property, we know there is an optimal solution OPT where x and y are siblings.
- Removing x and y from OPT and replacing their parents in OPT by z results in a better solution for the subproblem, contradiction

Encoding & Decoding

- Encoding:
  - Create a hashmap or dictionary or array (indexed by characters) to store bit code of each character
  - Scan each character in the text, look for its code, and concatenate
- Decoding:
  - Go left/right the tree representing prefix-free code according to the current bit
  - If leaf is reached, concatenate that character to current text

Thank You!