A	0110	J	11000000	S	1001
B	00010	K	101000	T	1000
C	01111	L	10101	U	0000
D	0010	M	00011	V	1100001
E	111	N	010	W	001101
F	101001	O	01110	X	1100000101
G	11001	P	110001	Y	110000011
H	00111	Q	1100000100	Z	001100
I	1101	R	1011		

CSCI 241

Lecture N
Huffman Coding

Announcements

Announcements

- Interested in helping out with some CS education research?
    - Looking for someone to do some work over break on the test cases for the programming assignments in this class. Email me or come talk to me if you're interested.

Goals

- Fill out course evaluations.
- Understand the basic idea behind Huffman Coding
- Ponder some coding interview questions.

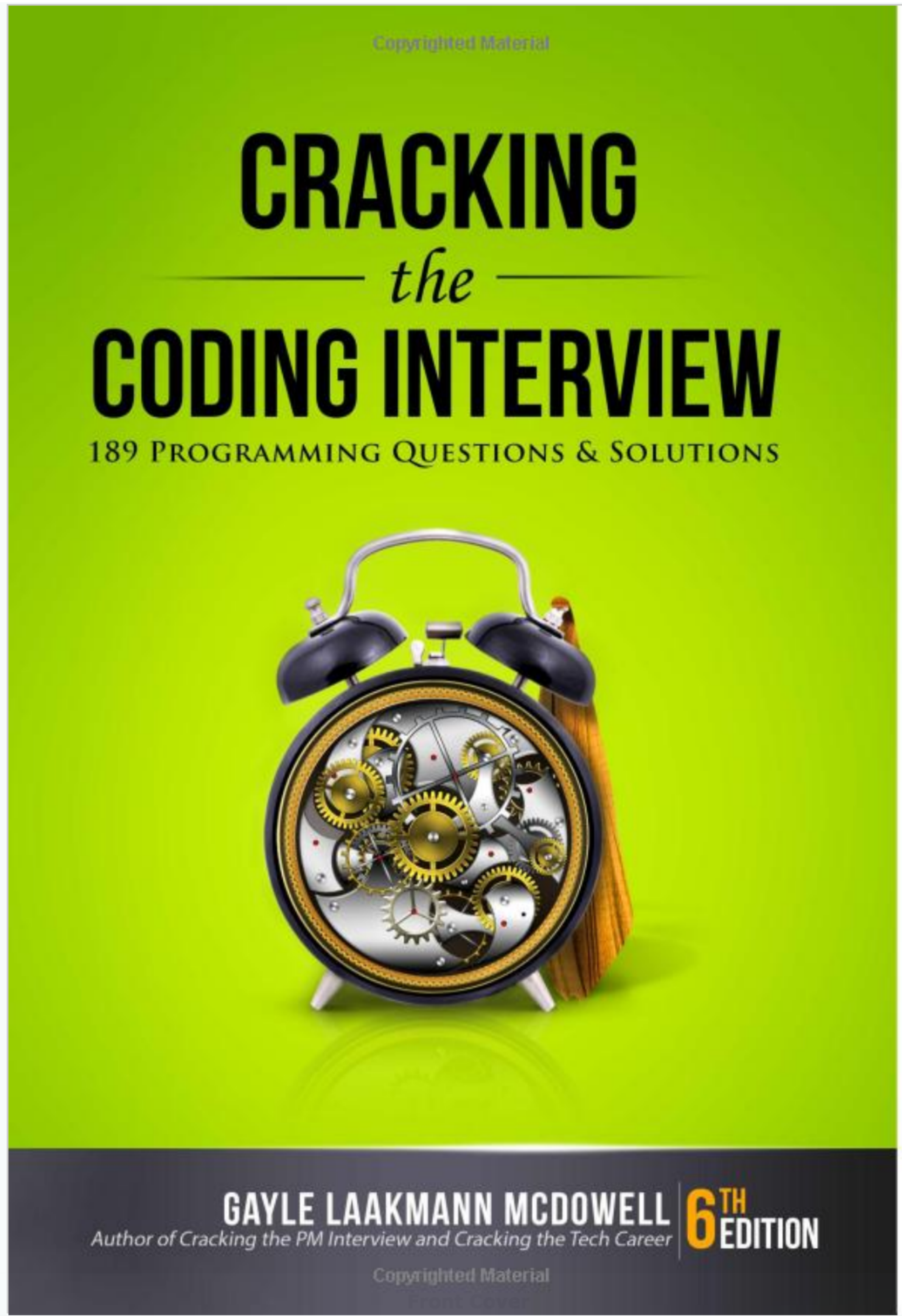
Course Evaluations

- Your feedback is helpful and I will read it carefully (after I submit grades).
- I’m teaching 241 again in Winter, so what you say will make a difference.

Practice Problems

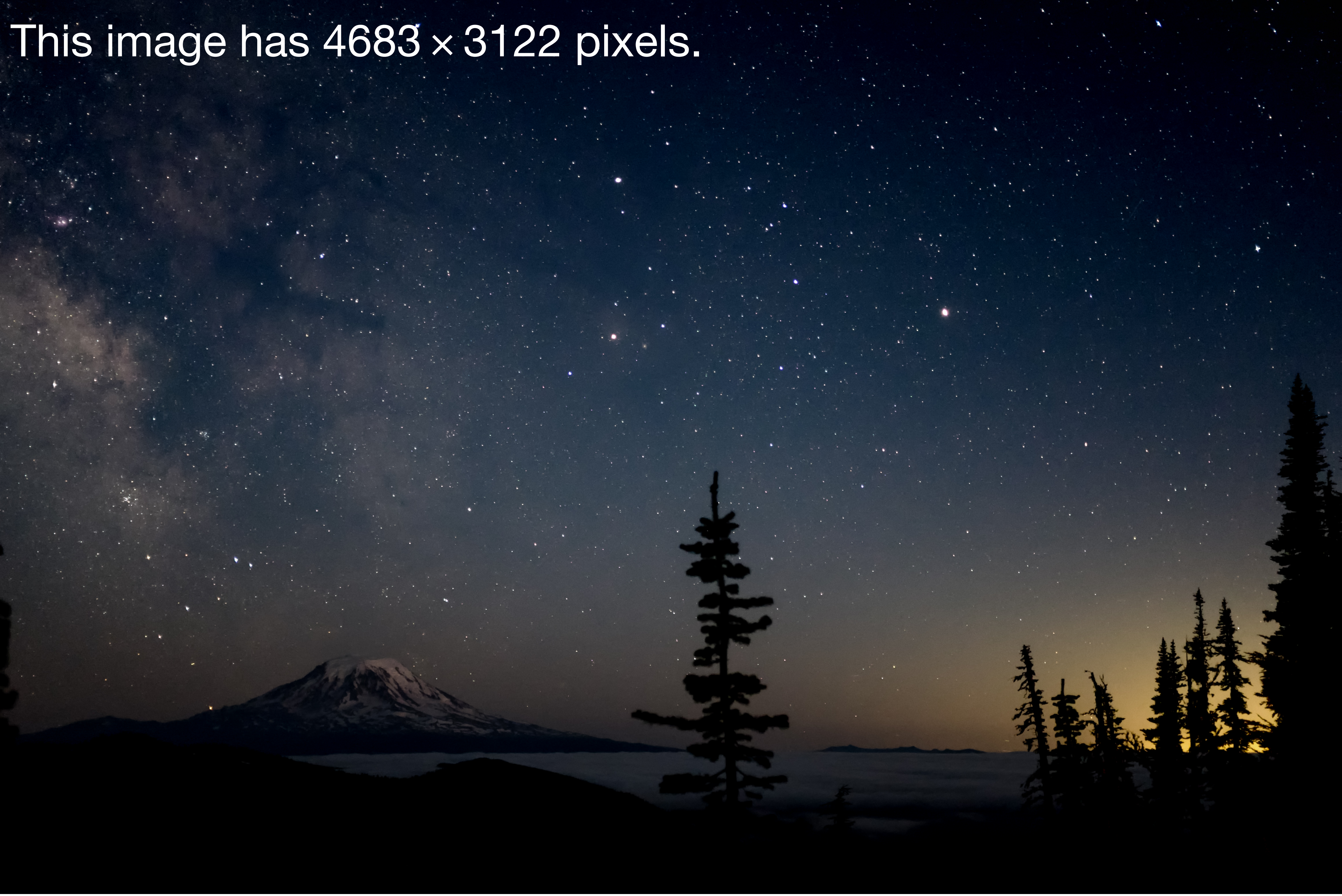
Practice Problems

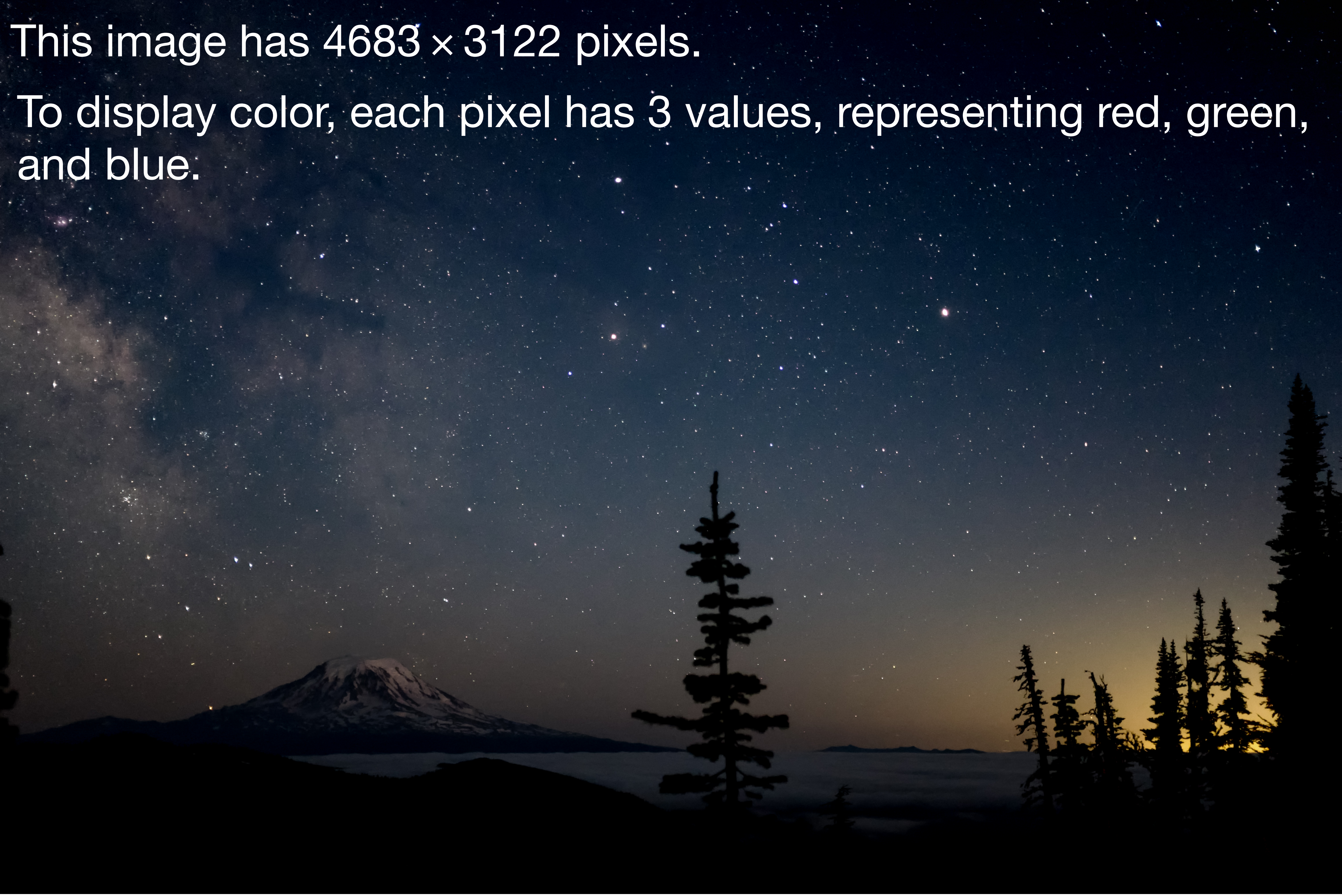
Practice Problems



Fun

- Easier: https://codingbat.com/java
    - good for recursion
- Easy -> Hard https://adventofcode.com/
    - see how far into December you can get before giving up
    - A couple years ago I got to ~Dec 21
- Easy -> Nuts https://projecteuler.net/
    - first 50 are pretty manageable
-

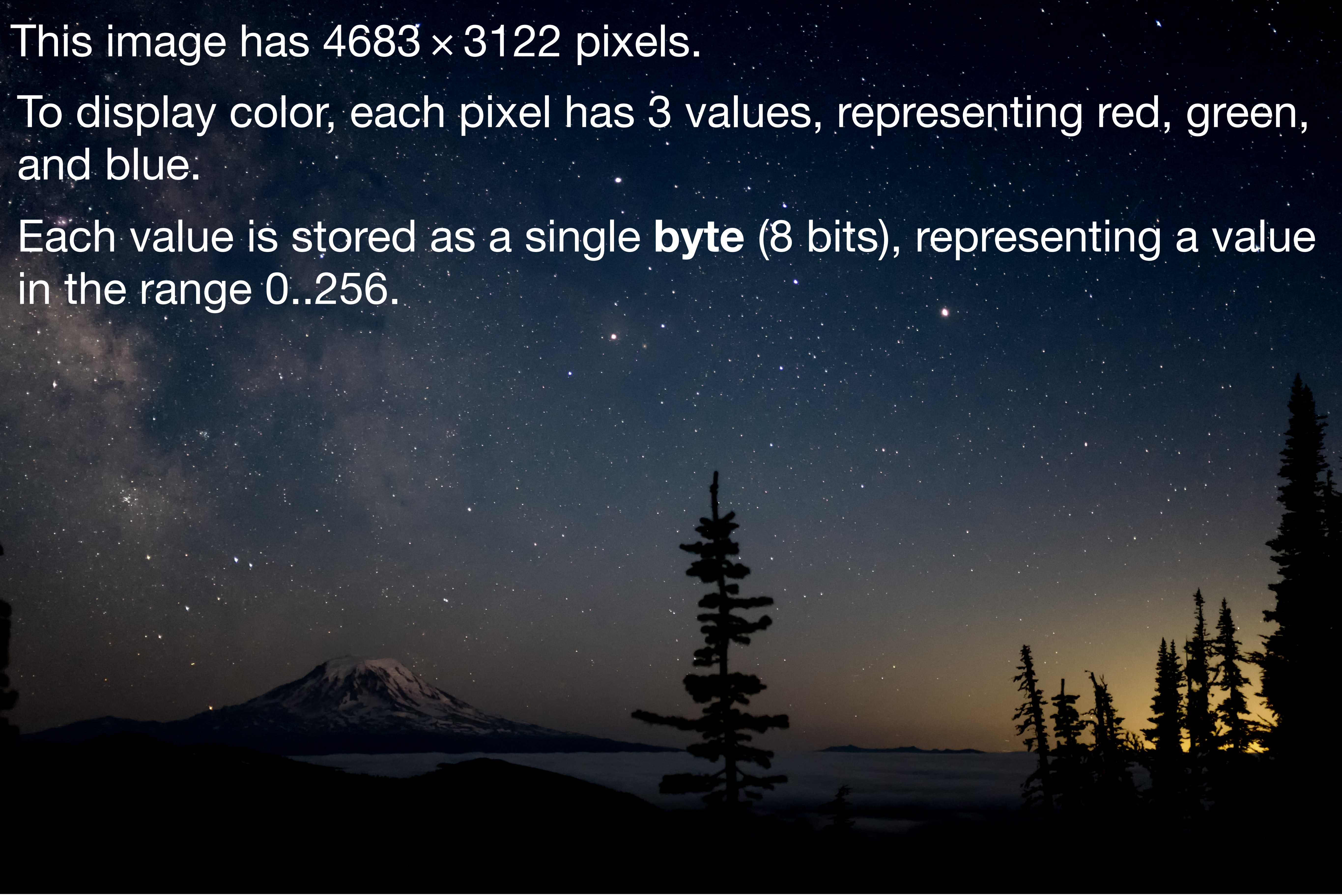

This image has 4683 × 3122 pixels.

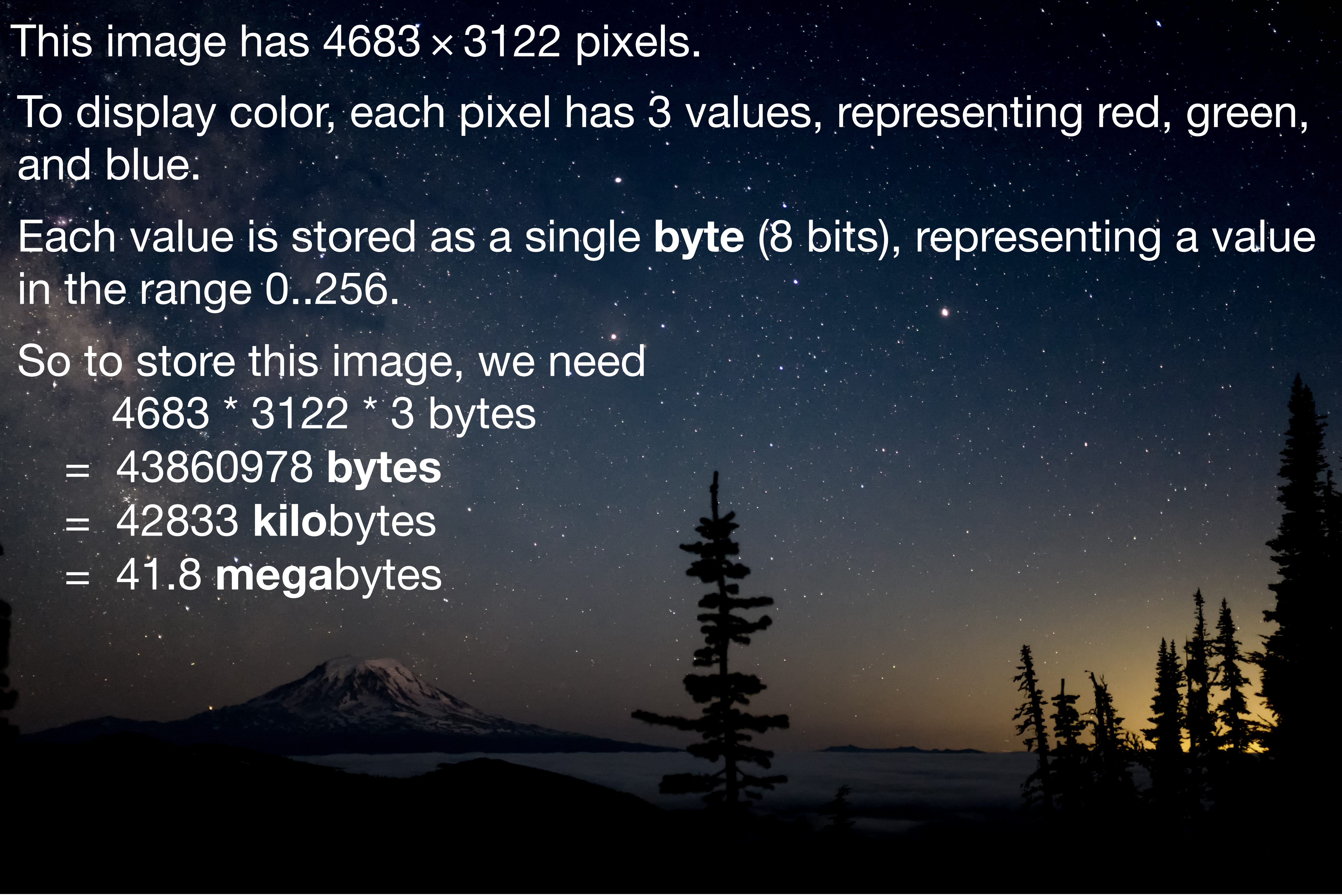

This image has 4683 × 3122 pixels.
To display color, each pixel has 3 values, representing red, green, and blue.

This image has 4683 × 3122 pixels.

To display color, each pixel has 3 values, representing red, green, and blue.

Each value is stored as a single **byte** (8 bits), representing a value in the range 0..256.





This image has 4683×3122 pixels.

To display color, each pixel has 3 values, representing red, green, and blue.

Each value is stored as a single **byte** (8 bits), representing a value in the range 0..256.

So to store this image, we need

4683 * 3122 * 3 bytes

= 43860978 **bytes**

= 42833 **kilo**bytes

= 41.8 **mega**bytes

This image has 4683×3122 pixels.

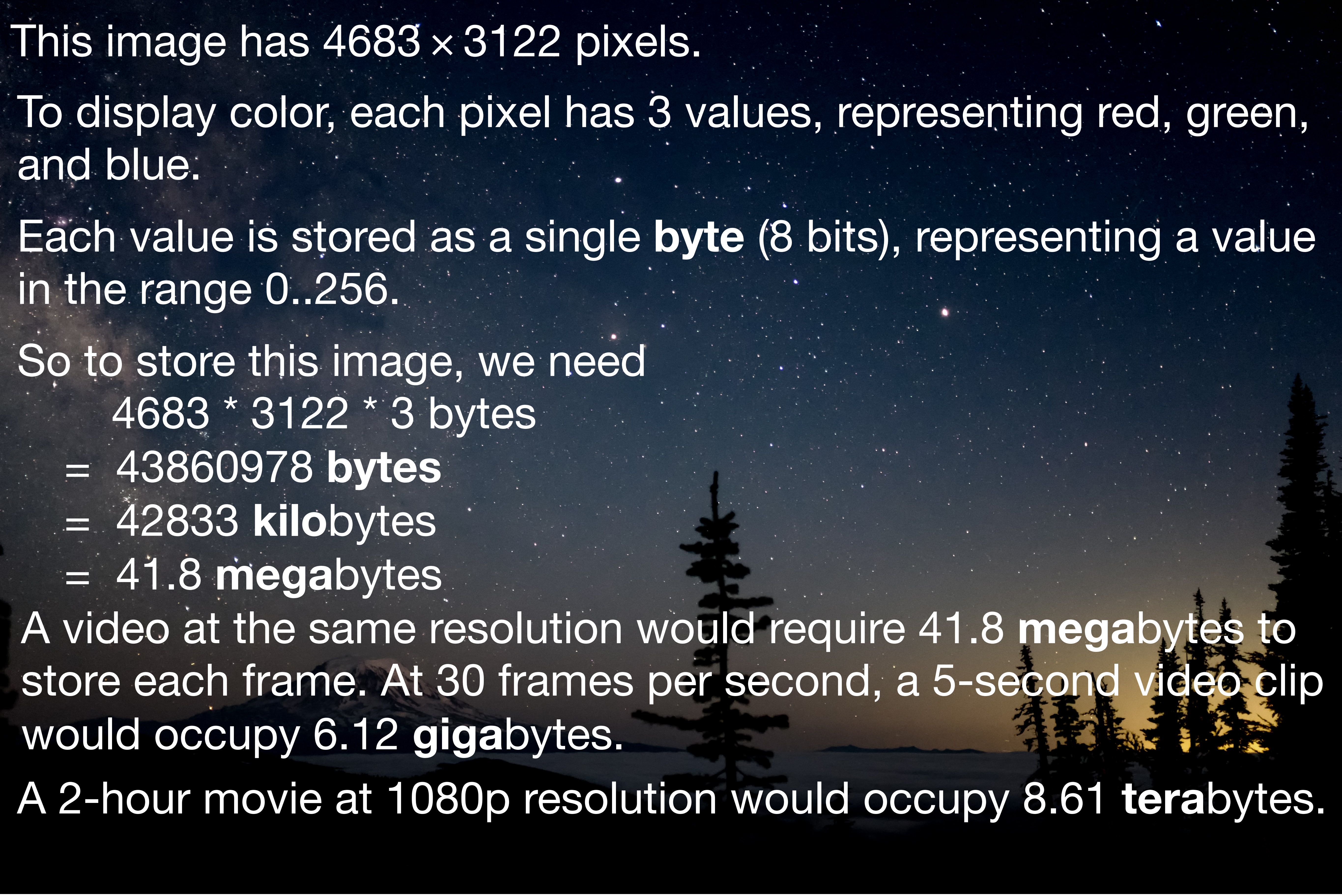
To display color, each pixel has 3 values, representing red, green, and blue.

Each value is stored as a single **byte** (8 bits), representing a value in the range 0..256.

So to store this image, we need

$$
\begin{aligned}
& 4683 * 3122 * 3 \text{ bytes} \\
= \; & 43860978 \textbf{ bytes} \\
= \; & 42833 \textbf{ kilo}\text{bytes} \\
= \; & 41.8 \textbf{ mega}\text{bytes}
\end{aligned}
$$

A video at the same resolution would require 41.8 **mega**bytes to store each frame. At 30 frames per second, a 5-second video clip would occupy 6.12 **giga**bytes.



This image has 4683 × 3122 pixels.

To display color, each pixel has 3 values, representing red, green, and blue.

Each value is stored as a single **byte** (8 bits), representing a value in the range 0..256.

So to store this image, we need

4683 * 3122 * 3 bytes

= 43860978 **bytes**

= 42833 **kilo**bytes

= 41.8 **mega**bytes

A video at the same resolution would require 41.8 **mega**bytes to store each frame. At 30 frames per second, a 5-second video clip would occupy 6.12 **giga**bytes.

A 2-hour movie at 1080p resolution would occupy 8.61 **tera**bytes.

This image has 4683×3122 pixels.

To display color, each pixel has 3 values, representing red, green, and blue.

Each value is stored as a single **byte** (8 bits), representing a value in the range 0..256.

So to store this image, we need

$$4683 * 3122 * 3 \text{ bytes}$$

= 43860978 **bytes**

= 42833 **kilo**bytes

= 41.8 **mega**bytes

A video at the same resolution would require 41.8 **mega**bytes to store each frame. At 30 frames per second, a 5-second video clip would occupy 6.12 **giga**bytes.

A 2-hour movie at 1080p resolution would occupy 8.61 **tera**bytes.

Streaming such a movie from Netflix would require 1.22 gigabytes per second of bandwidth. The fastest home internet connection money can buy is 125 megabytes per second.

This image has 4683 × 3122 pixels.

To display color, each pixel has 3 values, representing red, green, and blue.

Each value is stored as a single **byte** (8 bits), representing a value in the range 0..256.

So to store this image, we need

4683 * 3122 * 3 bytes

= 43860978 **bytes**

= 42833 **kilo**bytes

= 41.8 **mega**bytes

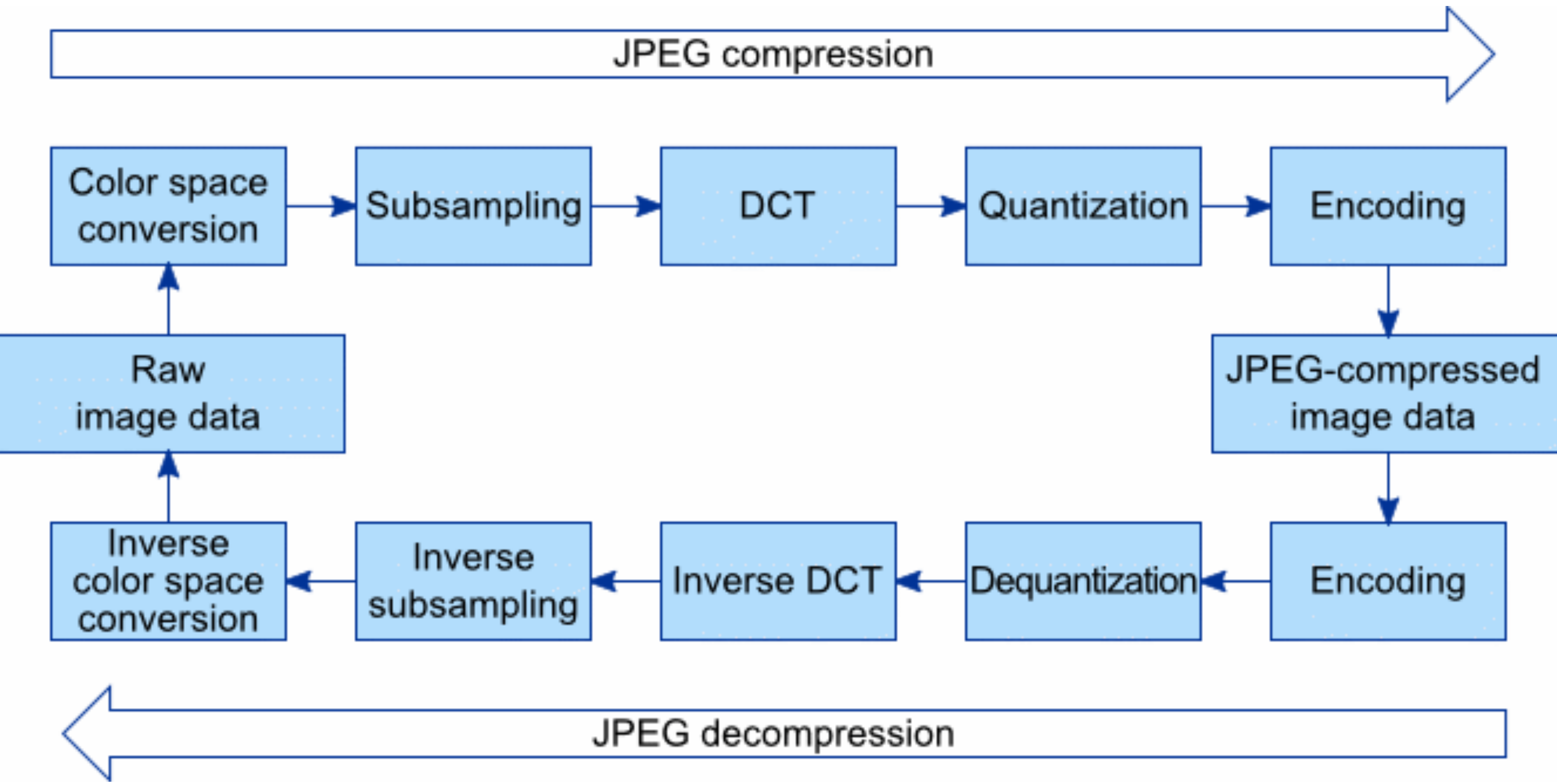
What gives?

A video at the same resolution would require 41.8 **mega**bytes to store each frame. At 30 frames per second, a 5-second video clip would occupy 6.12 **giga**bytes.

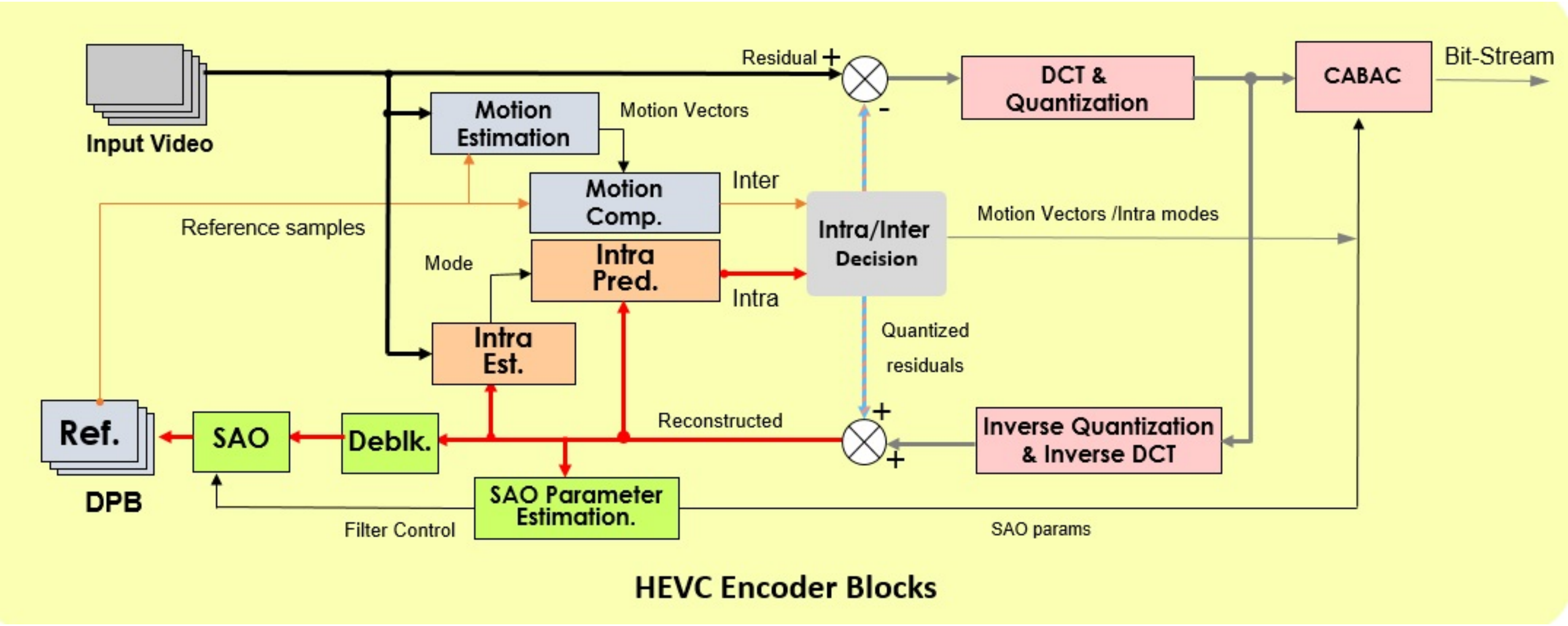
A 2-hour movie at 1080p resolution would occupy 8.61 **tera**bytes.

Streaming such a movie from Netflix would require 1.22 gigabytes per second of bandwidth. The fastest home internet connection money can buy is 125 megabytes per second.

Image Compression: JPEG



Video Compression: H.265



Video Compression: H.265

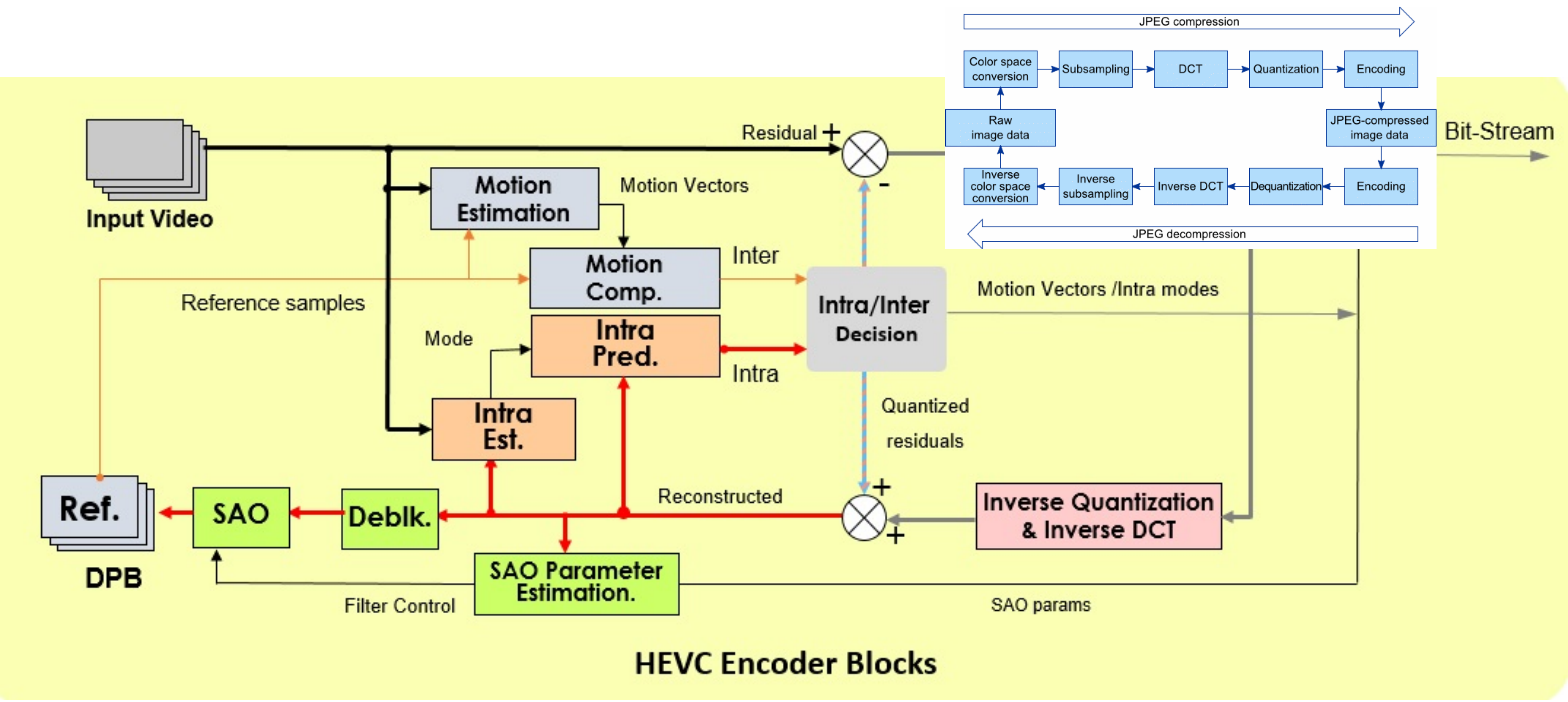
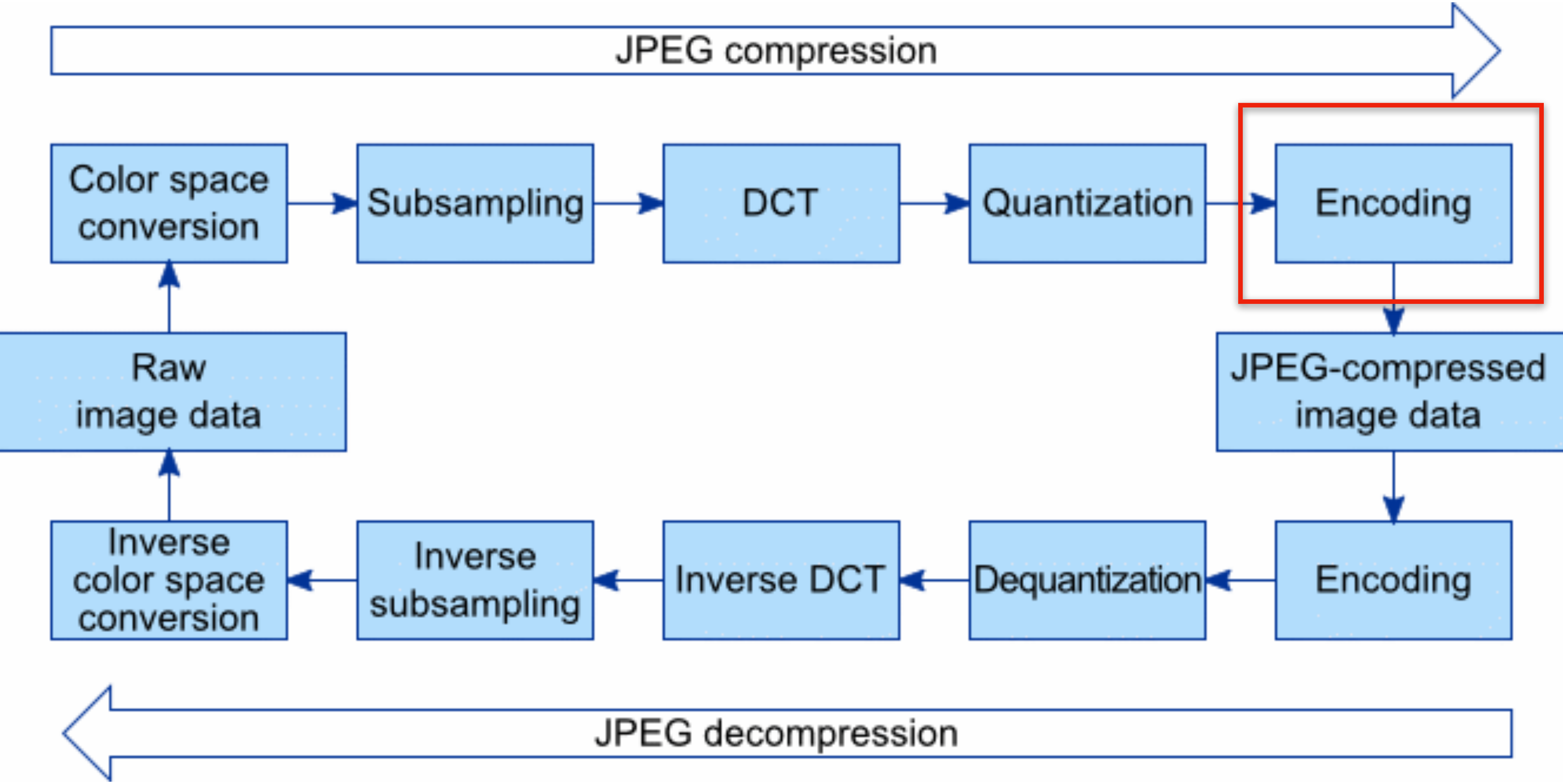


Image Compression: JPEG



A Method for the Construction of Minimum-Redundancy Codes*

DAVID A. HUFFMAN+, ASSOCIATE, IRE

Summary—An optimum method of coding an ensemble of messages consisting of a finite number of members is developed. A minimum-redundancy code is one constructed in such a way that the average number of coding digits per message is minimized.

Introduction

ONE IMPORTANT METHOD of transmitting messages is to transmit in their place sequences of symbols. If there are more messages which might be sent than there are kinds of symbols available, then some of the messages must use more than one symbol. If it is assumed that each symbol requires the same time for transmission, then the time for transmission (length) of a message is directly proportional to the number of symbols associated with it. In this paper, the symbol or sequence of symbols associated with a given message will be called the "message code." The entire number of messages which might be transmitted will be defined here as an ensemble code which, for a message ensemble consisting of a finite number of members, N, and for a given number of coding digits, D, yields the lowest possible average message length. In order to avoid the use of the lengthy term "minimum-redundancy," this term will be replaced here by "optimum." It will be understood then that, in this paper, "optimum code" means "minimum-redundancy code."

The following basic restrictions will be imposed on an ensemble code:

(a) No two messages will consist of identical arrangements of coding digits.

(b) The message codes will be constructed in such a way that no additional indication is necessary to specify where a message code begins and ends once the starting point of a sequence of messages is known.

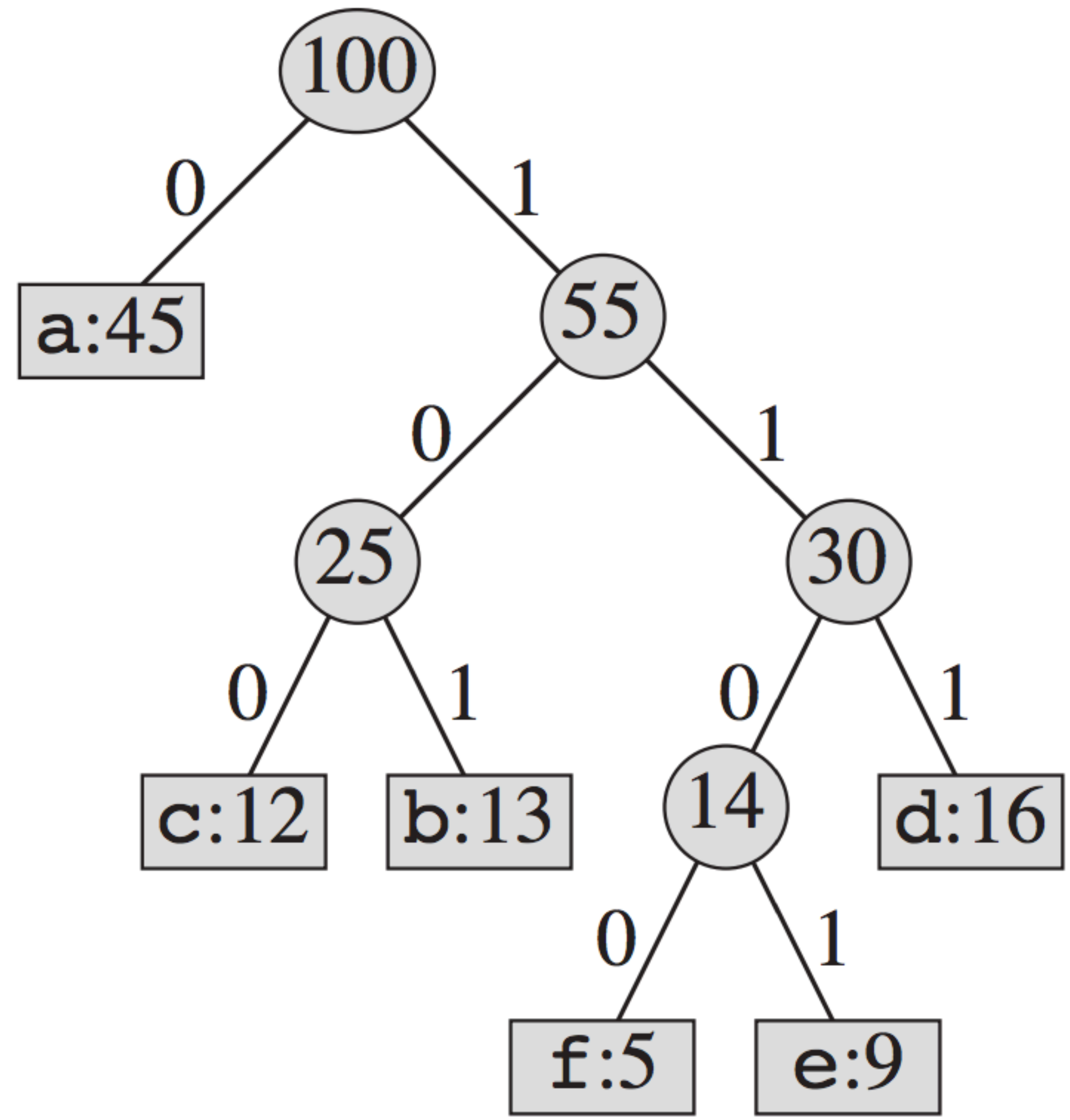
This is a coding tree.

Encodes a **map**ping from bit strings to words:
  0 means go left
  1 means go right

```
0: a

111: d

101: b
```



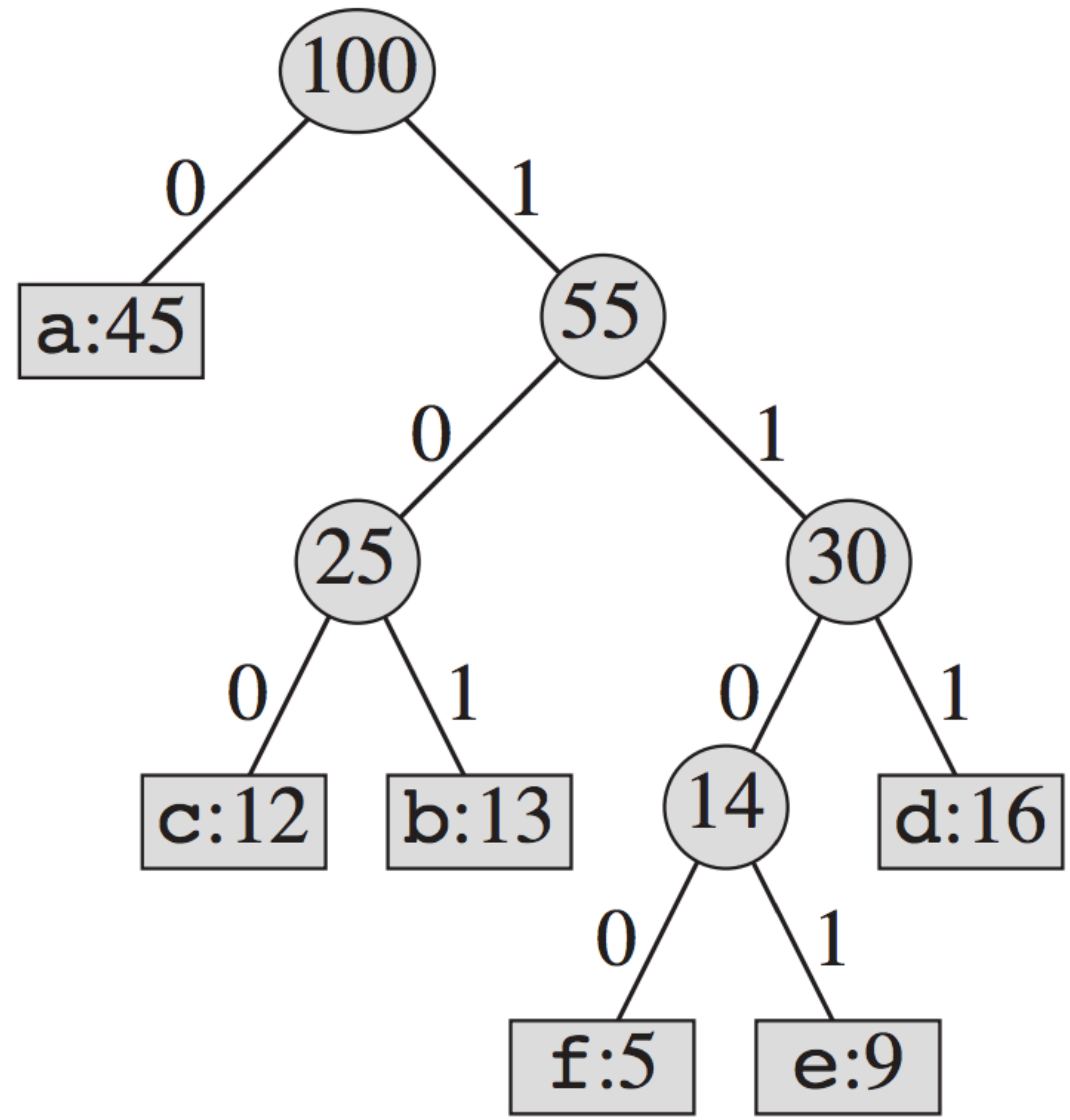
This is a coding tree.

Encodes a **map**ping from bit strings to words:
 0 means go left
 1 means go right

```
0: a

111: d

101: b
```

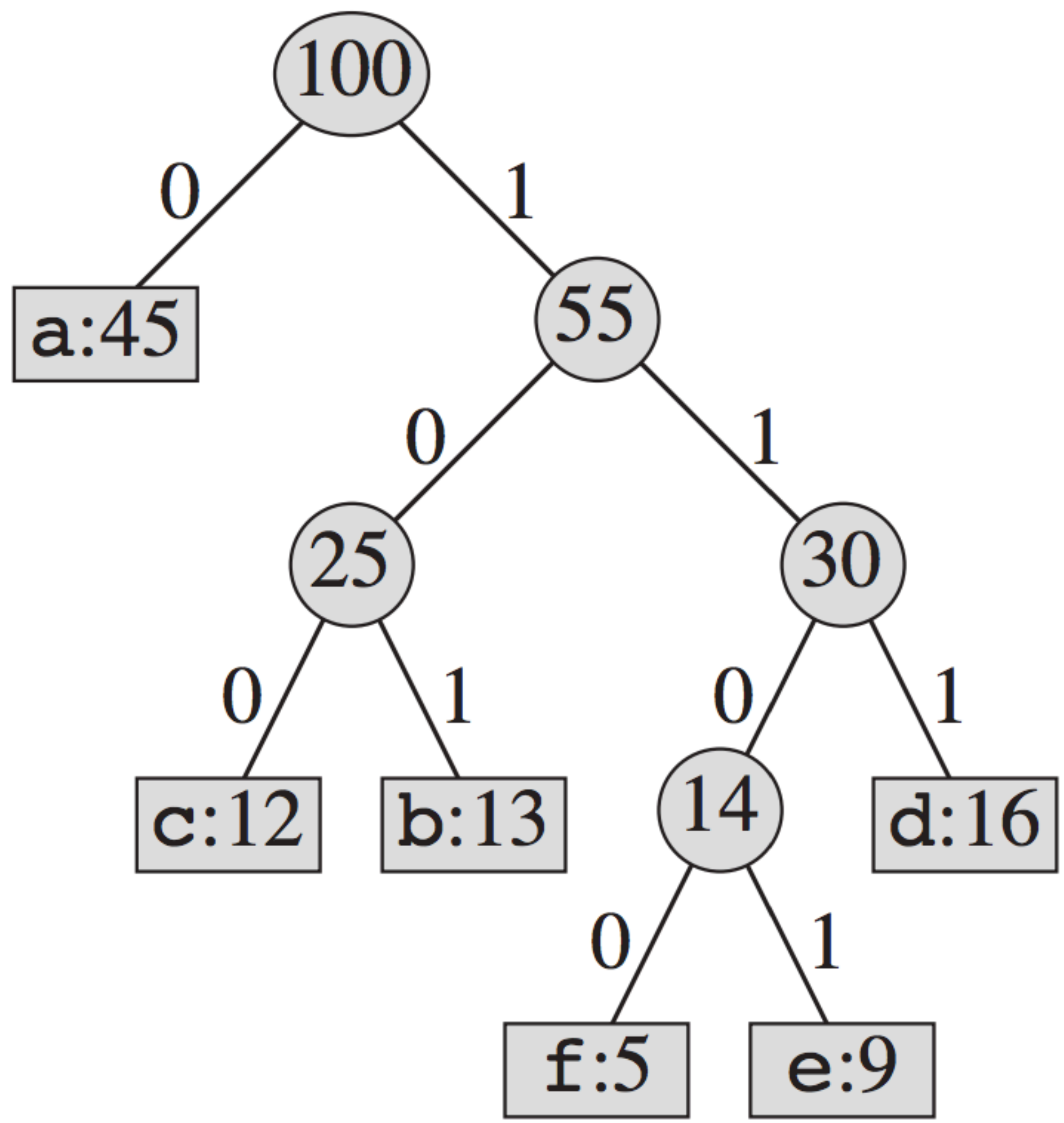


Key intuition: put common words near the root.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n - 1
    allocate a new node z
    z.left = x = EXTRACT-MIN(Q)
    z.right = y = EXTRACT-MIN(Q)
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)    // return the root of the tree
```

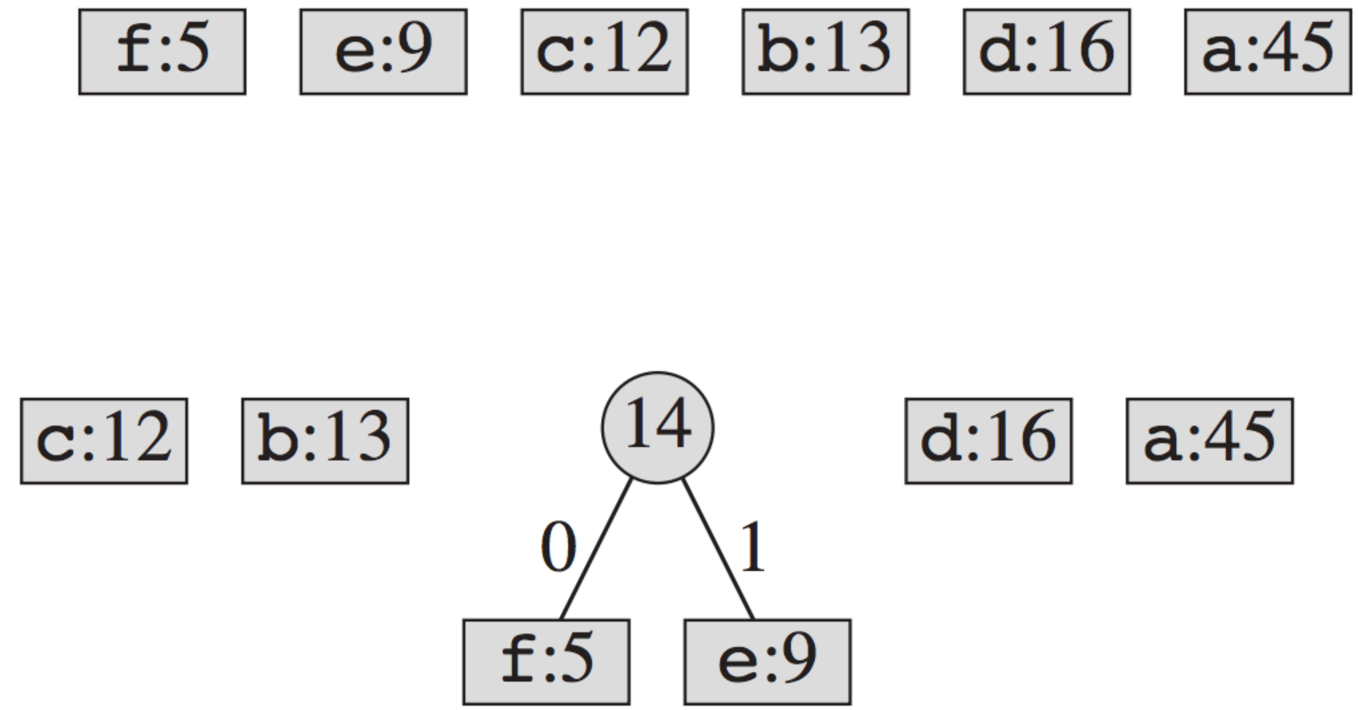
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Variable-length codeword	0	101	100	111	1101	1100




f:5
e:9
c:12
b:13
d:16
a:45

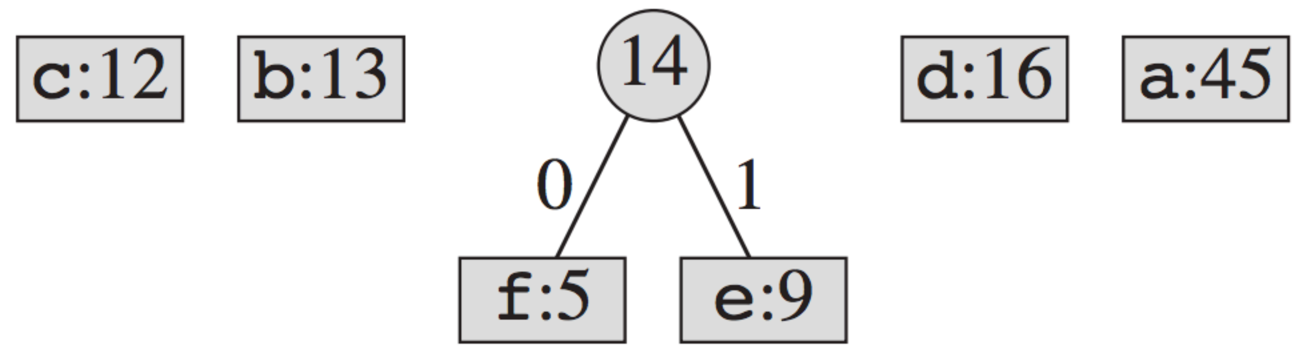
Smallest two: 5, 9

Smallest two: 5, 9

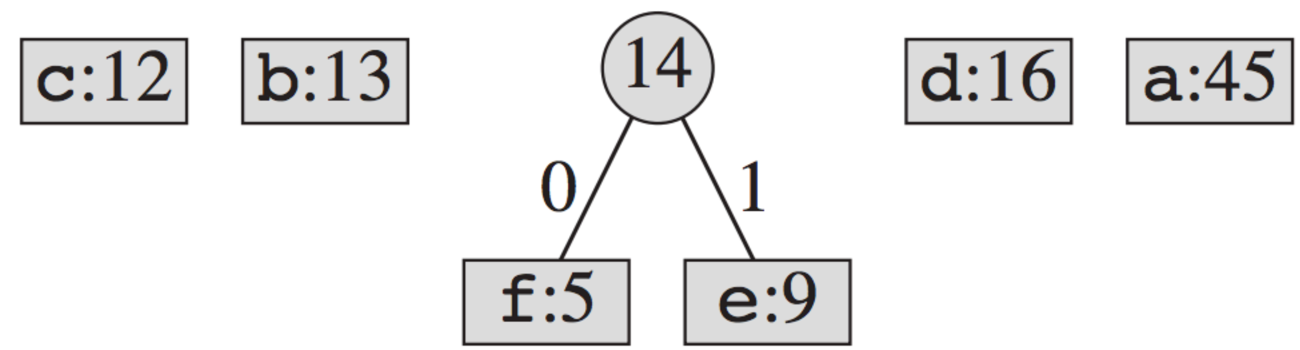


Smallest two: 5, 9

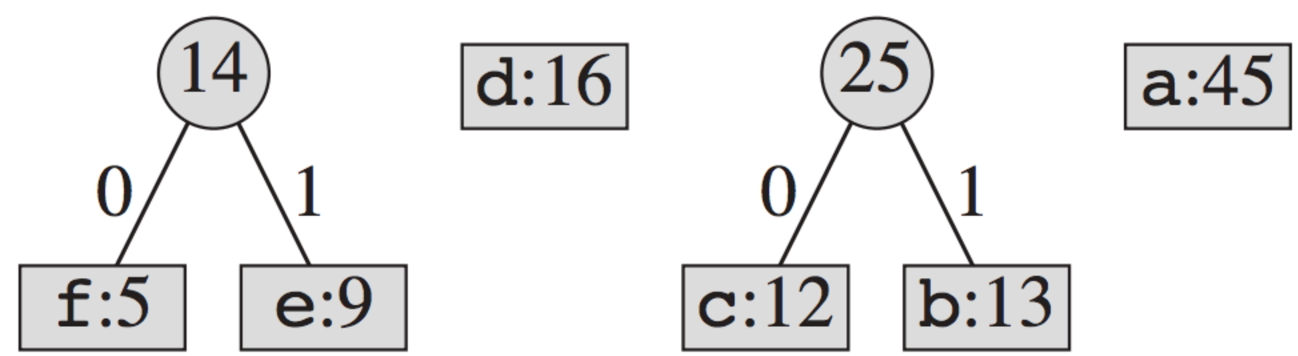
Smallest two: 12, 13



Smallest two: 5, 9

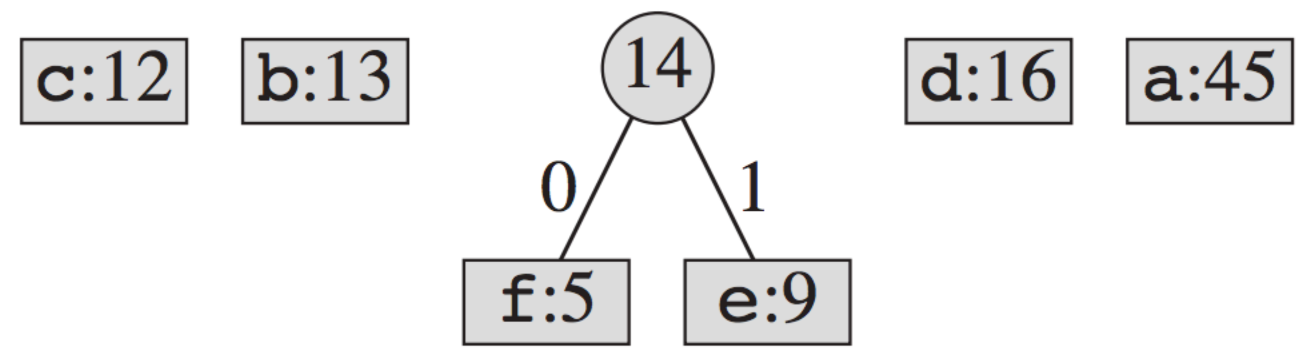


Smallest two: 12, 13

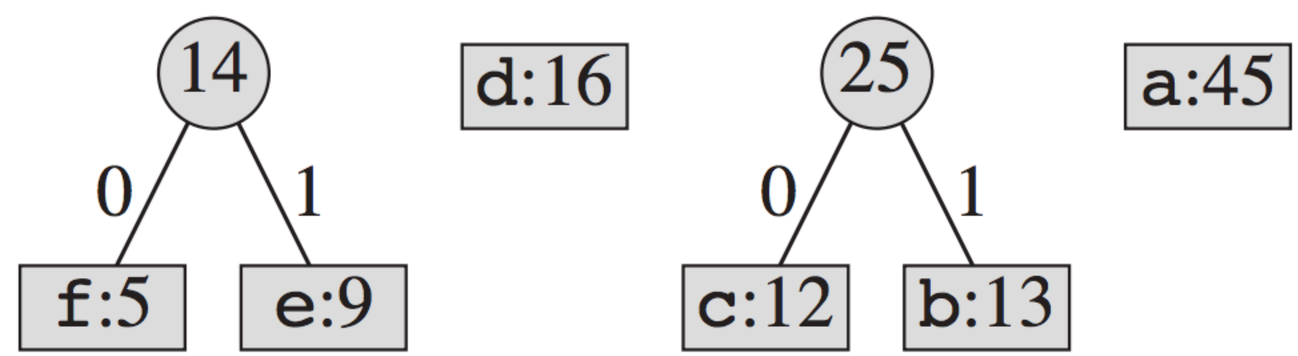


Smallest two: 5, 9

Smallest two: 12, 13

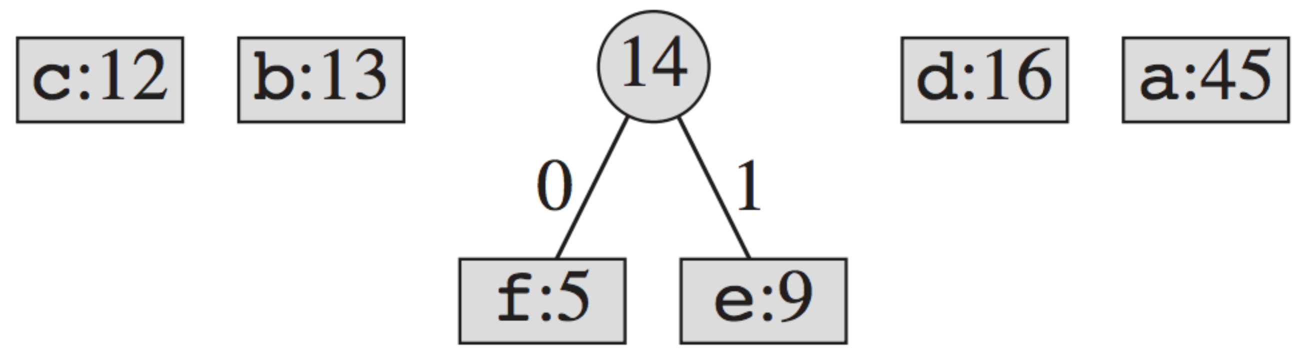


Smallest two: 14, 16

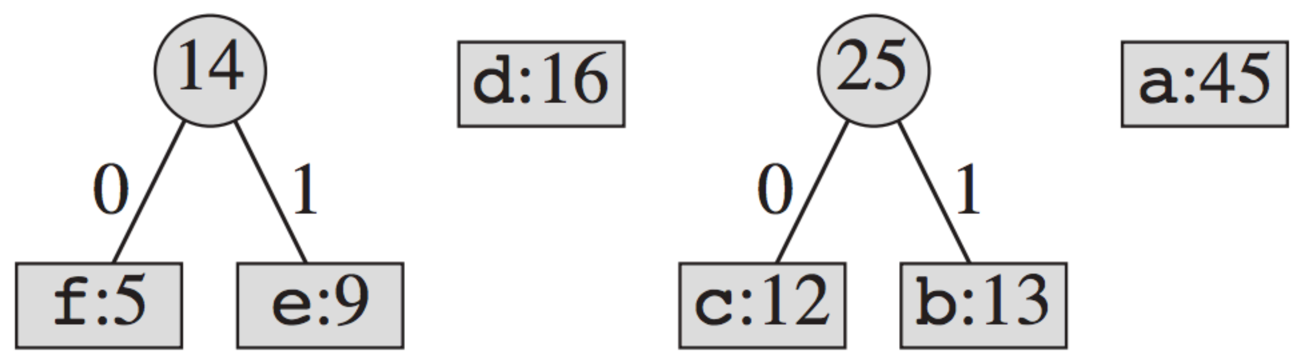


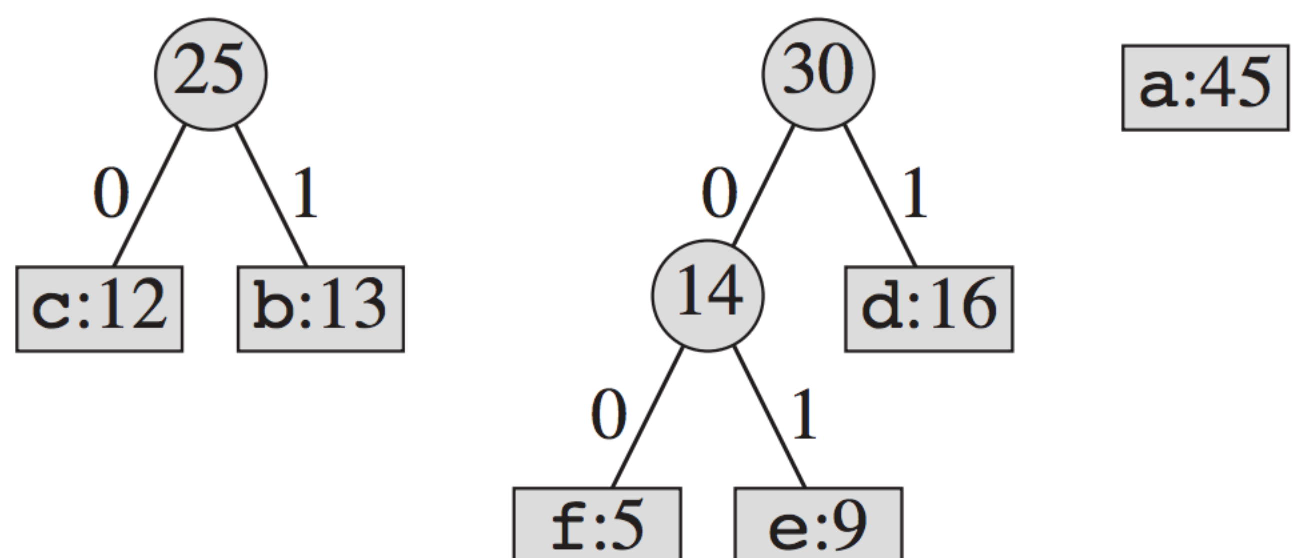
Smallest two: 5, 9

Smallest two: 12, 13



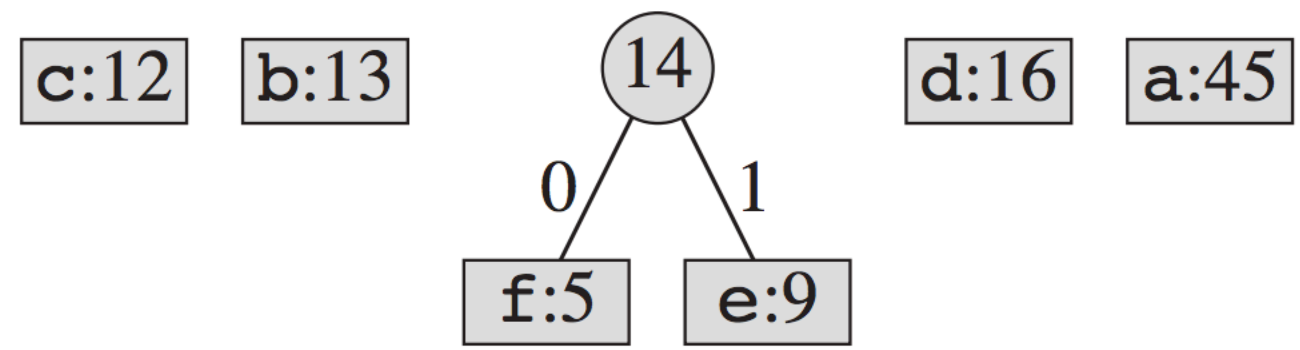
Smallest two: 14, 16



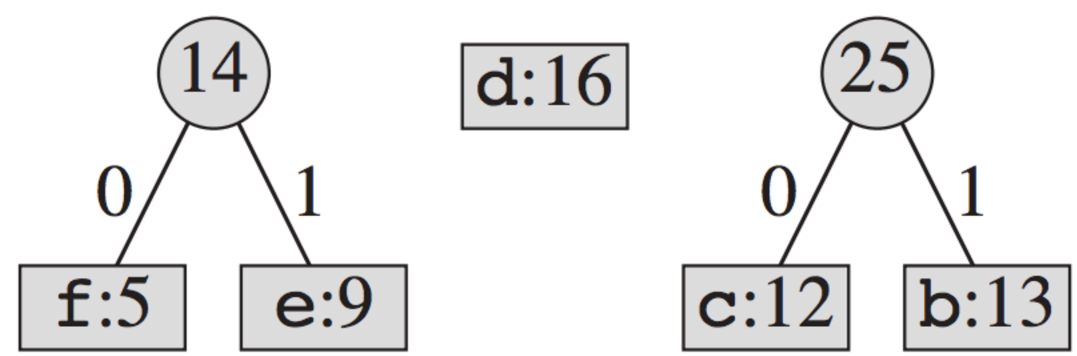


Smallest two: 5, 9

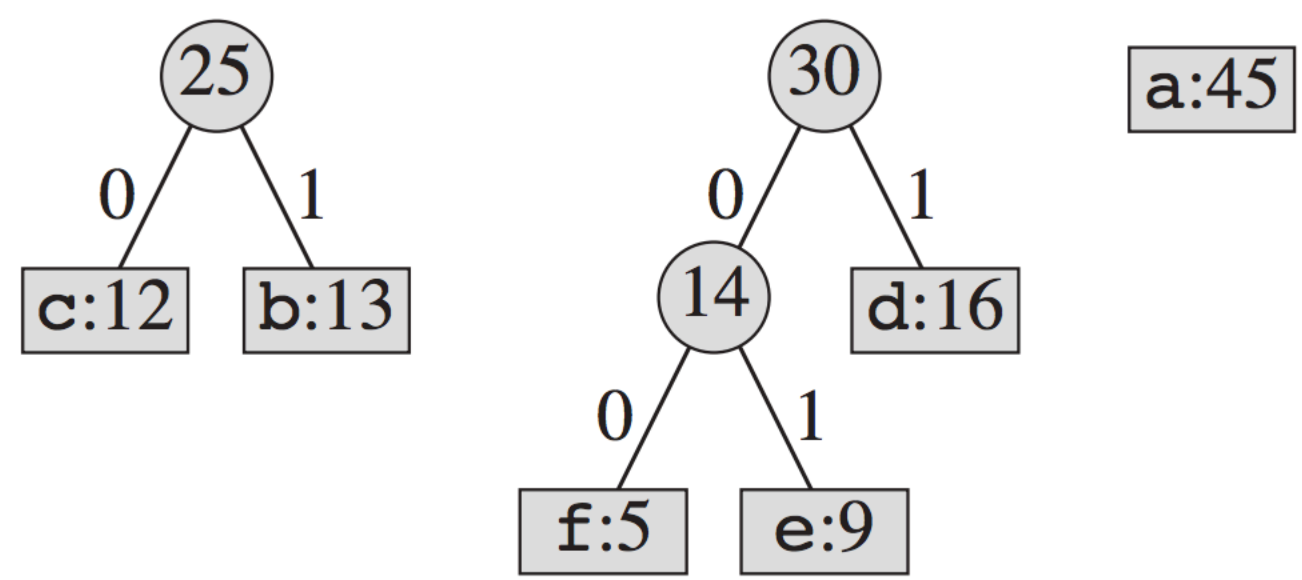
Smallest two: 12, 13

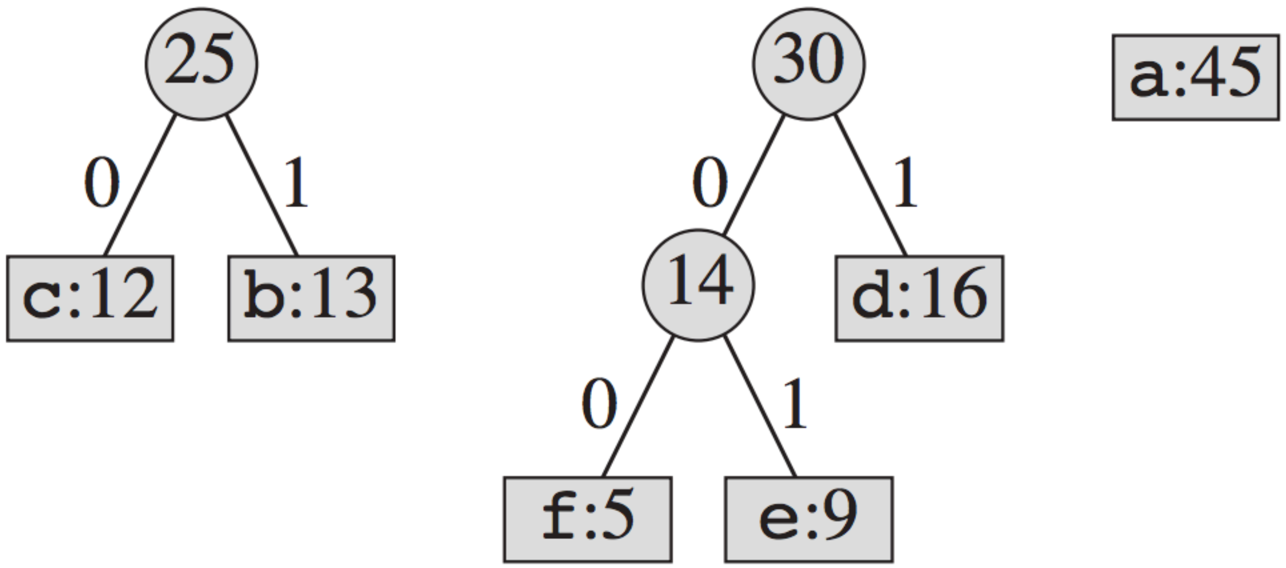


Smallest two: 14, 16

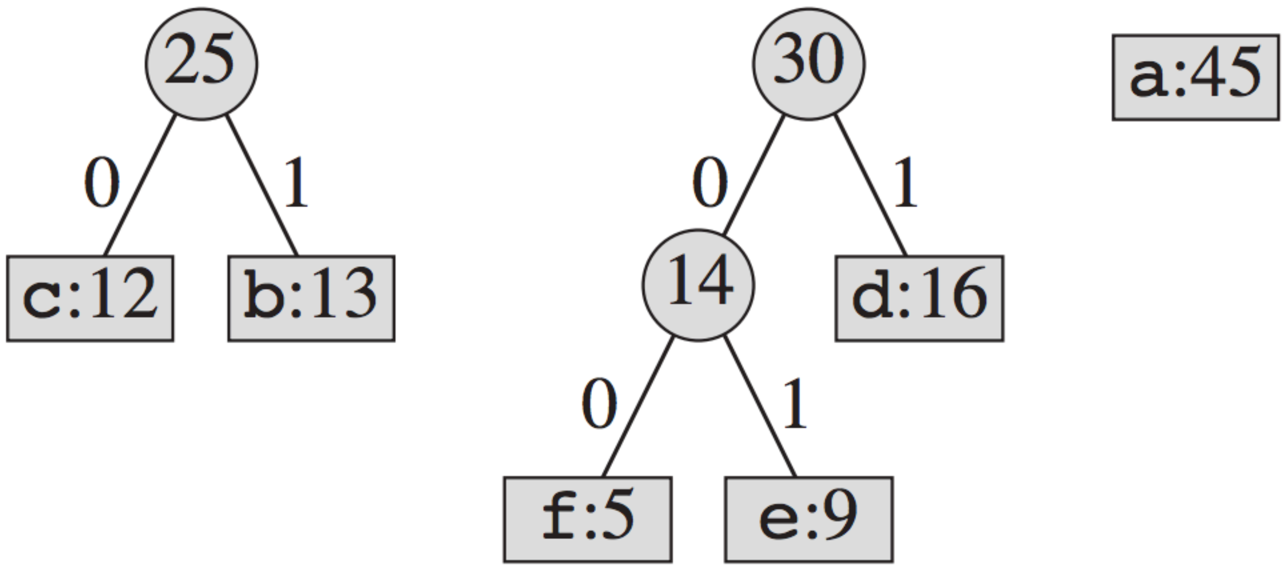


Smallest two: 25, 30

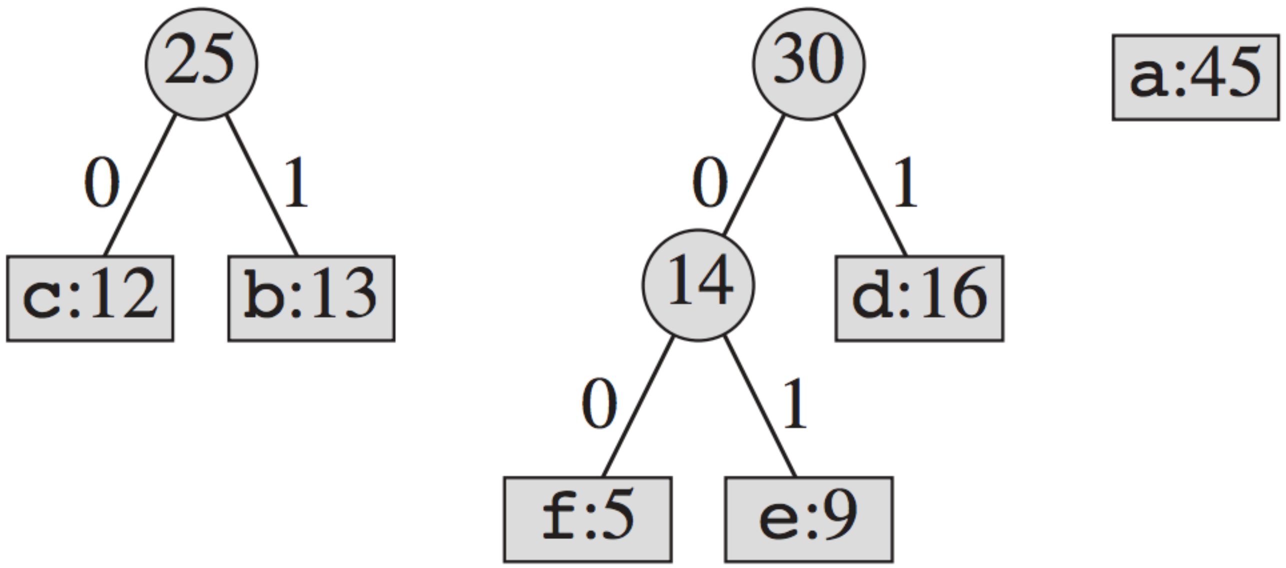



25
0
1
c:12
b:13
30
0
1
14
d:16
0
1
f:5
e:9
a:45

Smallest two: 25, 30

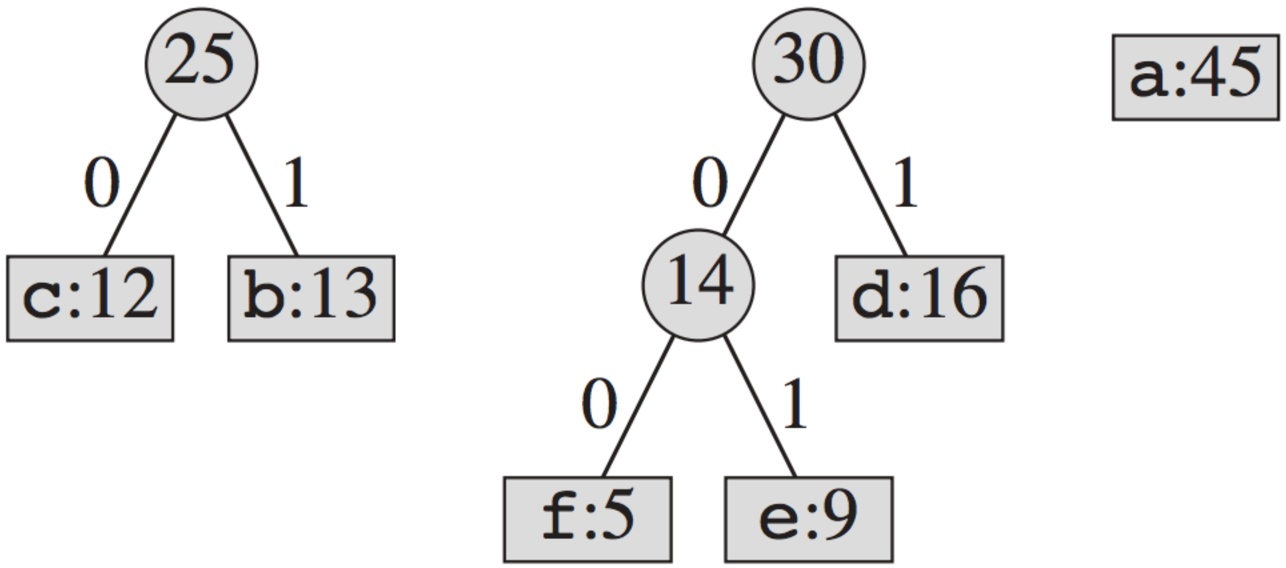


Smallest two: 25, 30

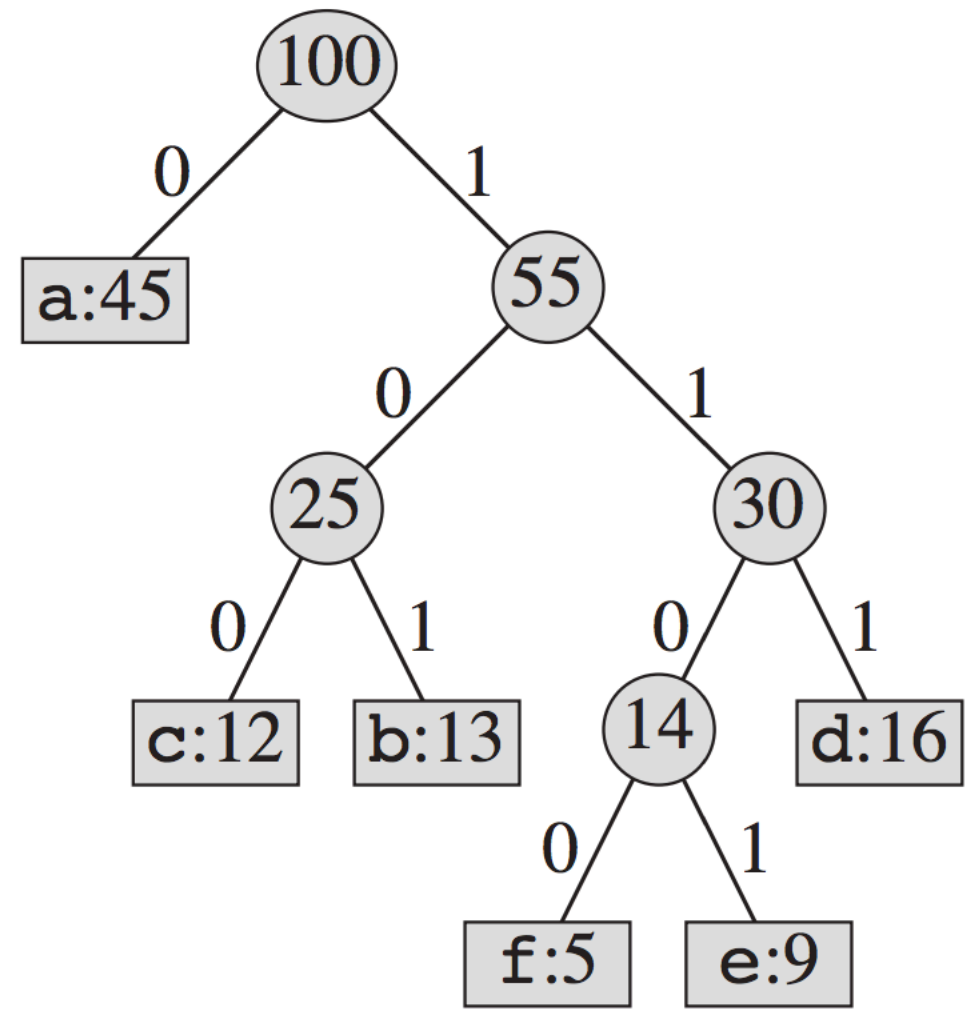


Huffman Tree:

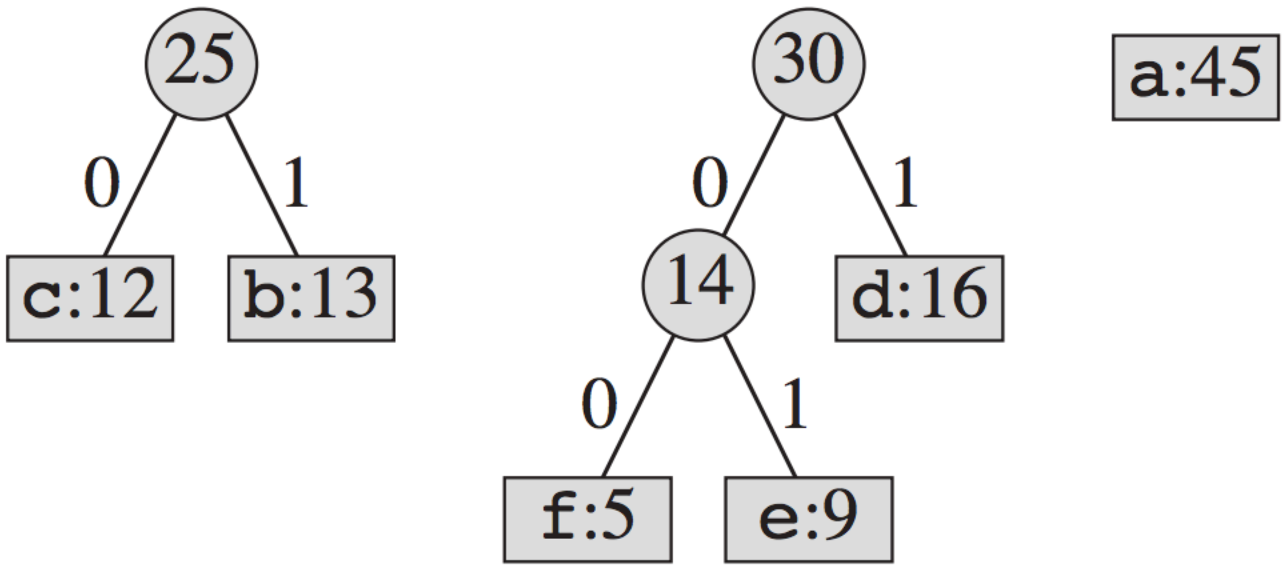
Smallest two: 25, 30

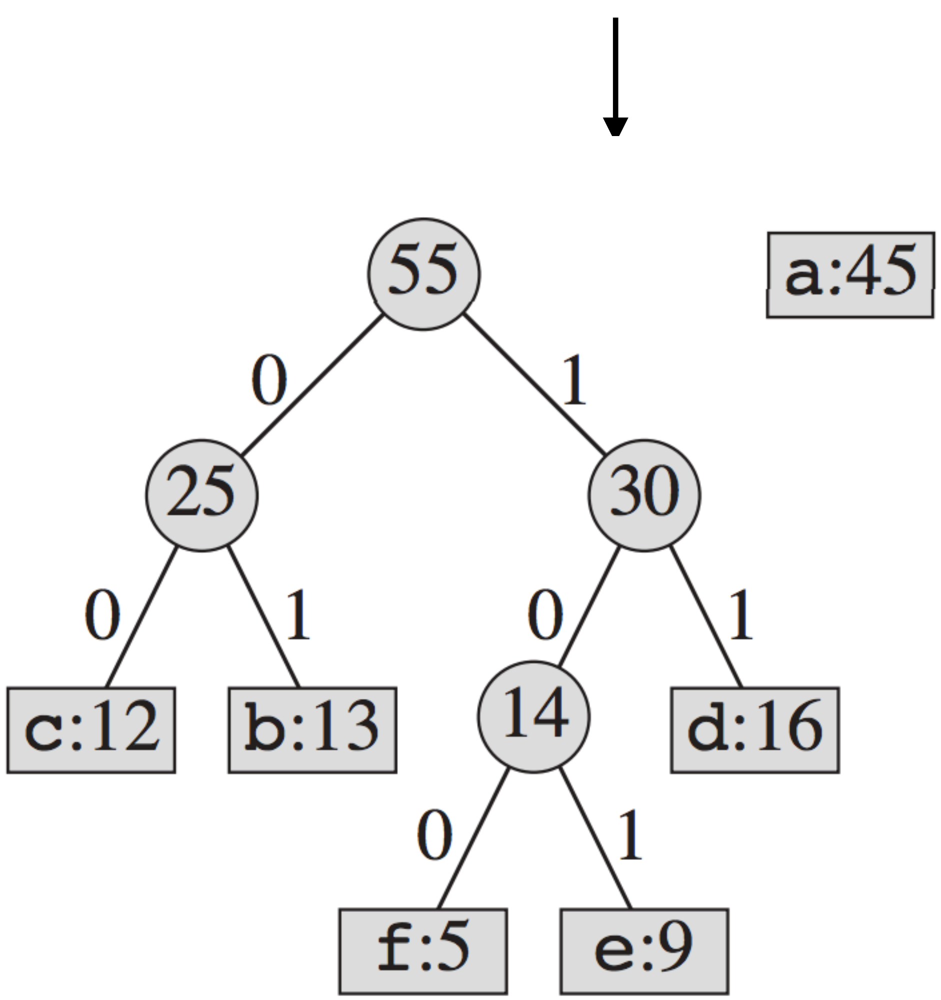


Huffman Tree:

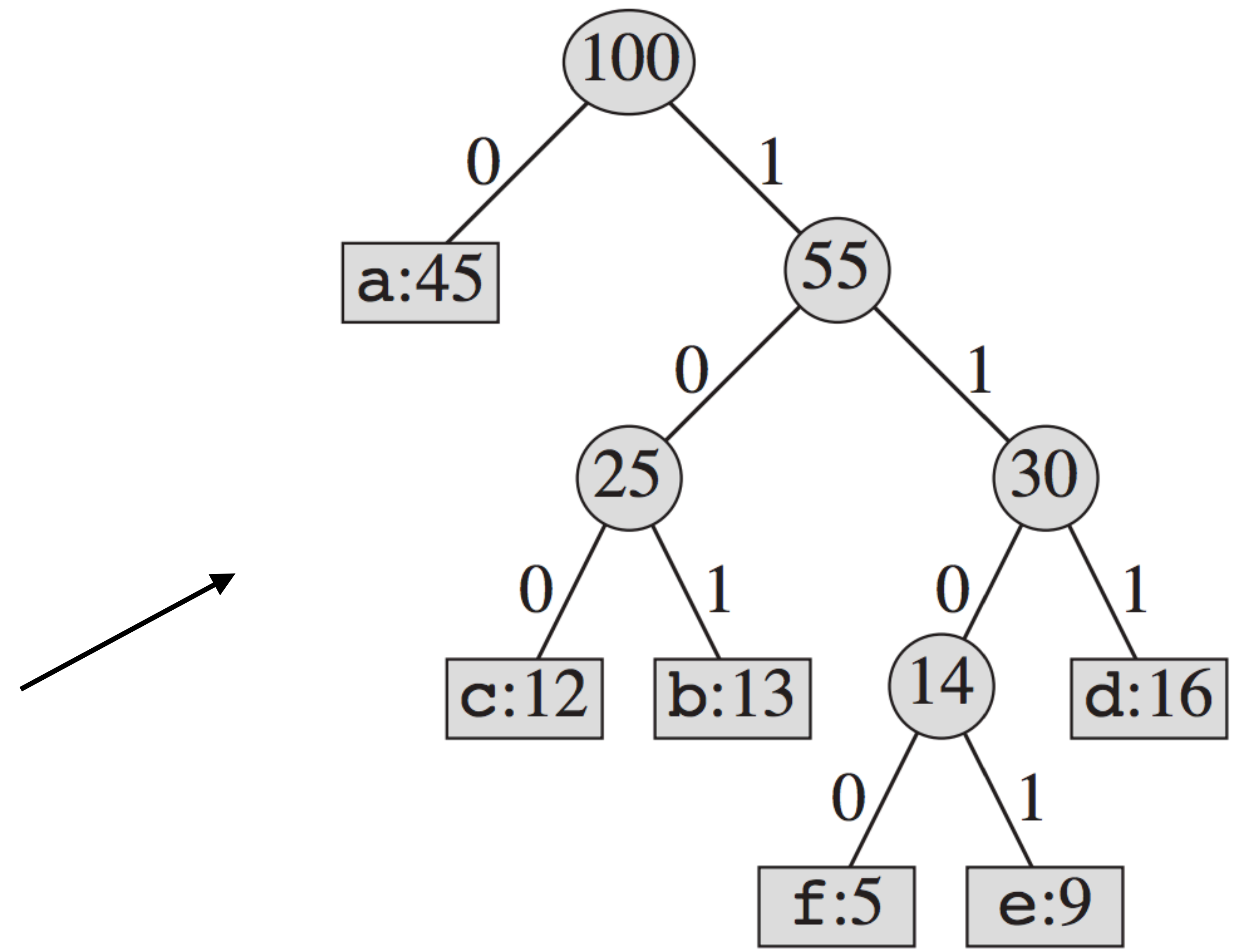


Smallest two: 25, 30





Huffman Tree:



Coding Huffman Coding