CS 241: Systems Programming
Lecture 31. Huffman Compression

Spring 2020
Prof. Stephen Checkoway

Announcements

Homework 6 is available

- Due 2020-05-13 at 11:00
- Late days cannot be used because I cannot accept any work after the end of the allotted time for finals

Poll on Piazza for when you want the final project due (May 13 is currently winning)

Reminder: There is no final, just the last assignment

Data representation

Data representation

Must have some way of representing information in computers

Computers are binary, so…

Data representation

Must have some way of representing information in computers

Computers are binary, so…

Binary Representation!

Number of bits required for text

a-zA-Z	52	(26 each upper and lower case)
0-9	20	(10 + shift characters)
other	22	(11 keys and shifted forms)
ws	5	(space, tab, lf, cr, vtab)
TOTAL:	99	

Need ceil($\log_2$ 99) => 7 bits per character

Character encodings

ASCII — 7 bit

- American national Standard Code for Information Interchange

ISO 8859-1 (Latin-1) — 8-bit code

- Uses ASCII for first half

Unicode — code points in the range 0–0x10FFFF

- UTF-32 — Fixed-length, 32-bit code units
- UTF-16 — Variable-length, one or two 16-bit code units per code point
- UTF-8 — Variable-length, 1–4 8-bit code units per code point
  - When most significant bit is 0, matches ASCII

Data Compression

Idea: reduce the number of bytes needed to represent data

100,000,000,000,000,000,000

100 Quintillion

1*10^20

1e20

Lossless Compression

Same information, but with different representation

All information can be recovered

Vs. "Lossy" compression like JPG or MP3

Example – small text file

Assume data with only the letters A-G

- need 3 bits to encode data (represent)

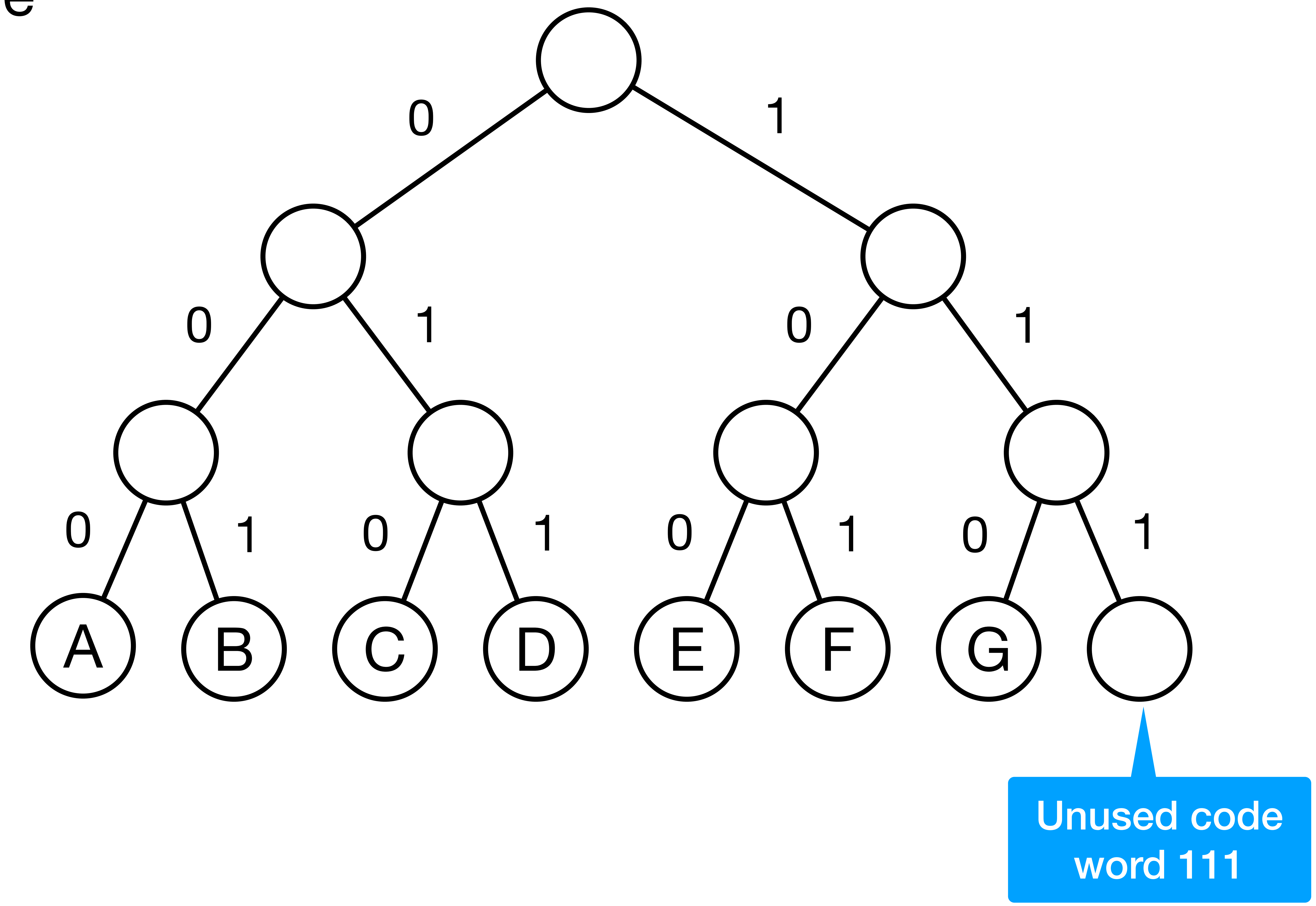
Letter	Bit rep	Count	Bits used
A	000	13	39
B	001	12	36
C	010	10	30
D	011	5	15
E	100	3	9
F	101	1	3
G	110	1	3

Total length: 39+36+30+15+9+3+3 = 135

Representing codes as a trie

Represent code using binary trie

- Binary tree
- Values only in leaves
- 0 is left, 1 is right



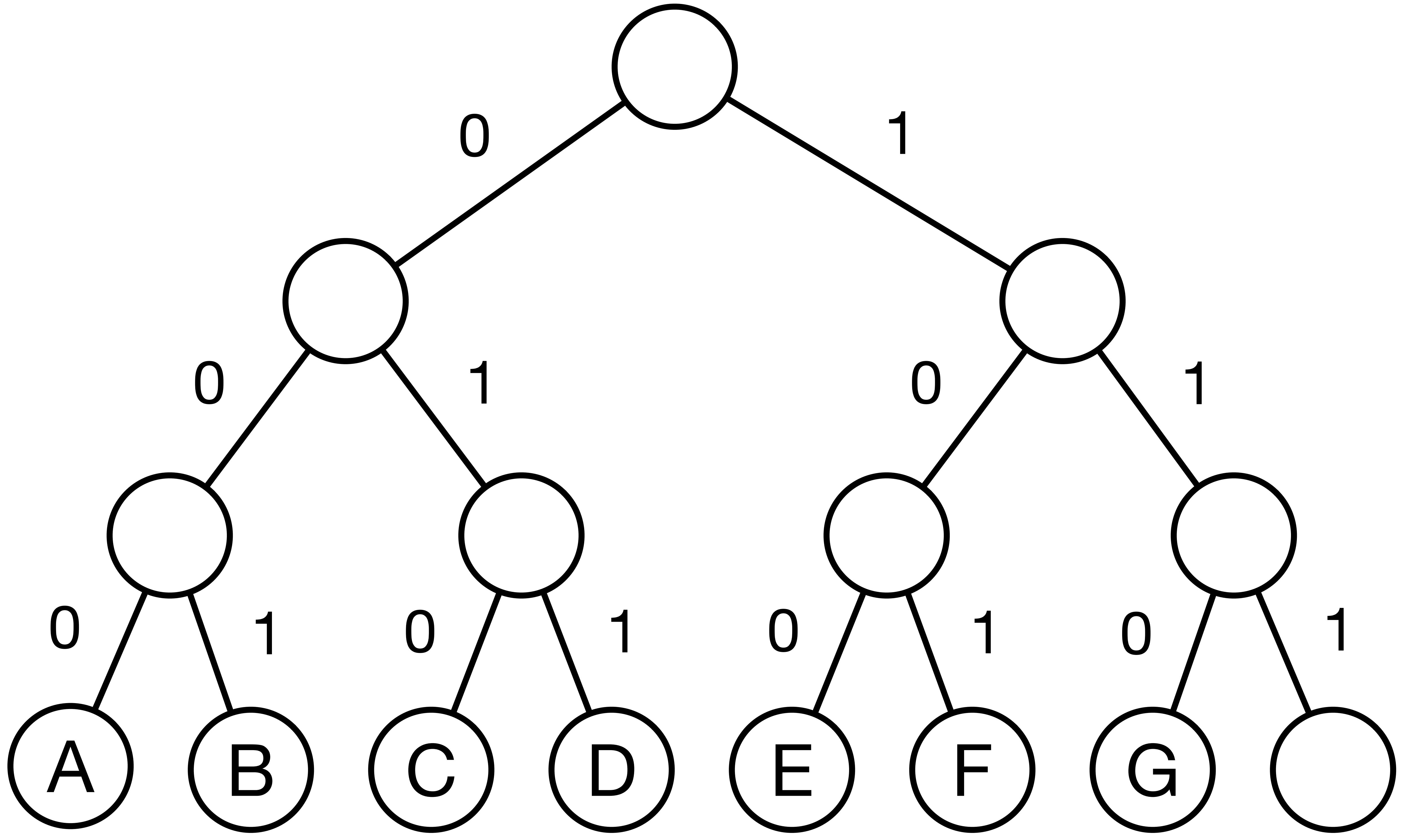
Encoding

To encode a character, walk the path from the root to the leaf

- Each time you go left, output a 0 bit
- Each time you go right, output a 1 bit

Encode example

How do we encode FED?



Decoding

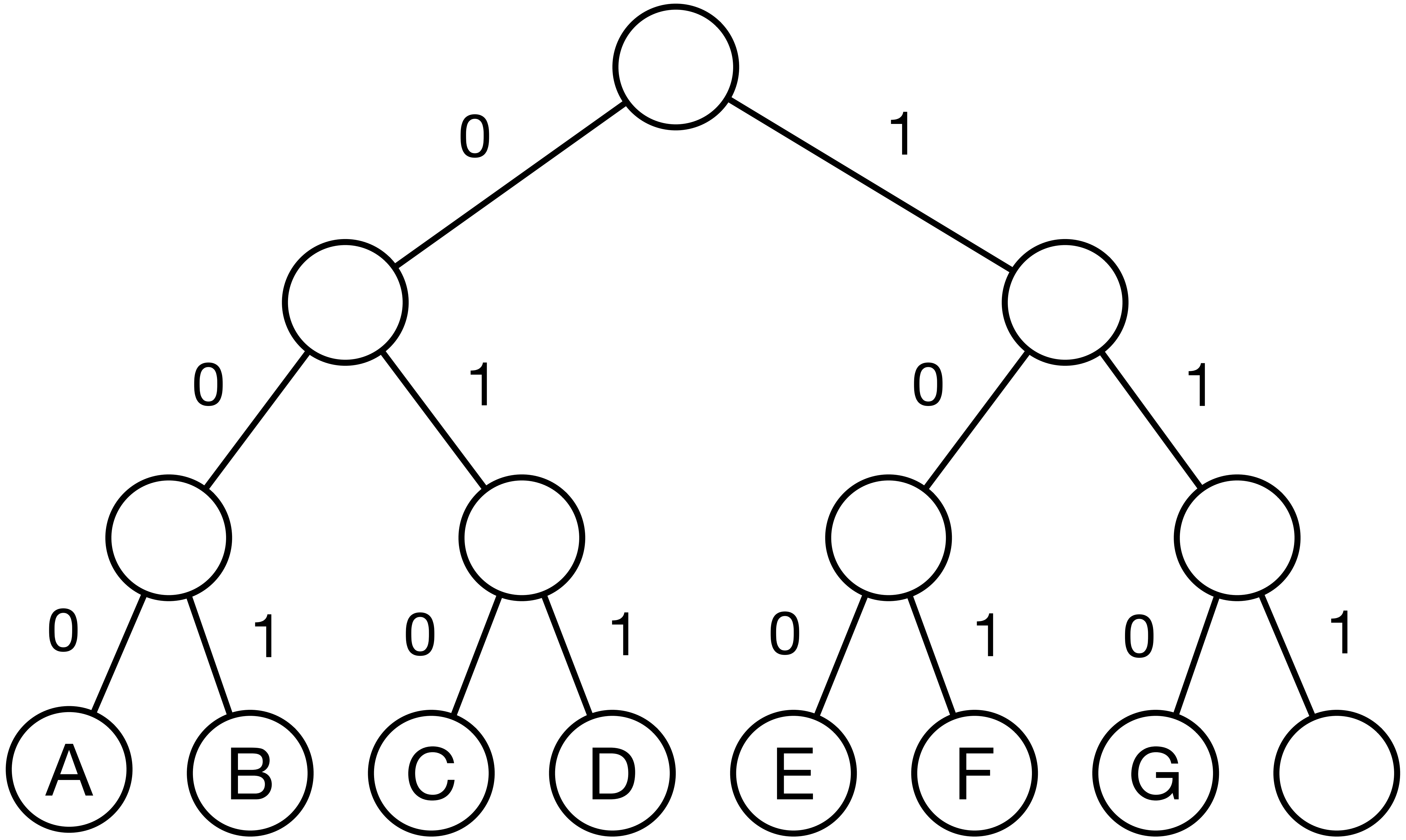
To decode a character, use the bits to choose which child to take, starting from the root

- If the current node is a leaf, output the corresponding character
- If the next bit is a 0, move to the left child
- If the next bit is a 1, move to the right child

Decode example

How do we decode
001100011?

What about 000111?



Desirable properties

Full tree

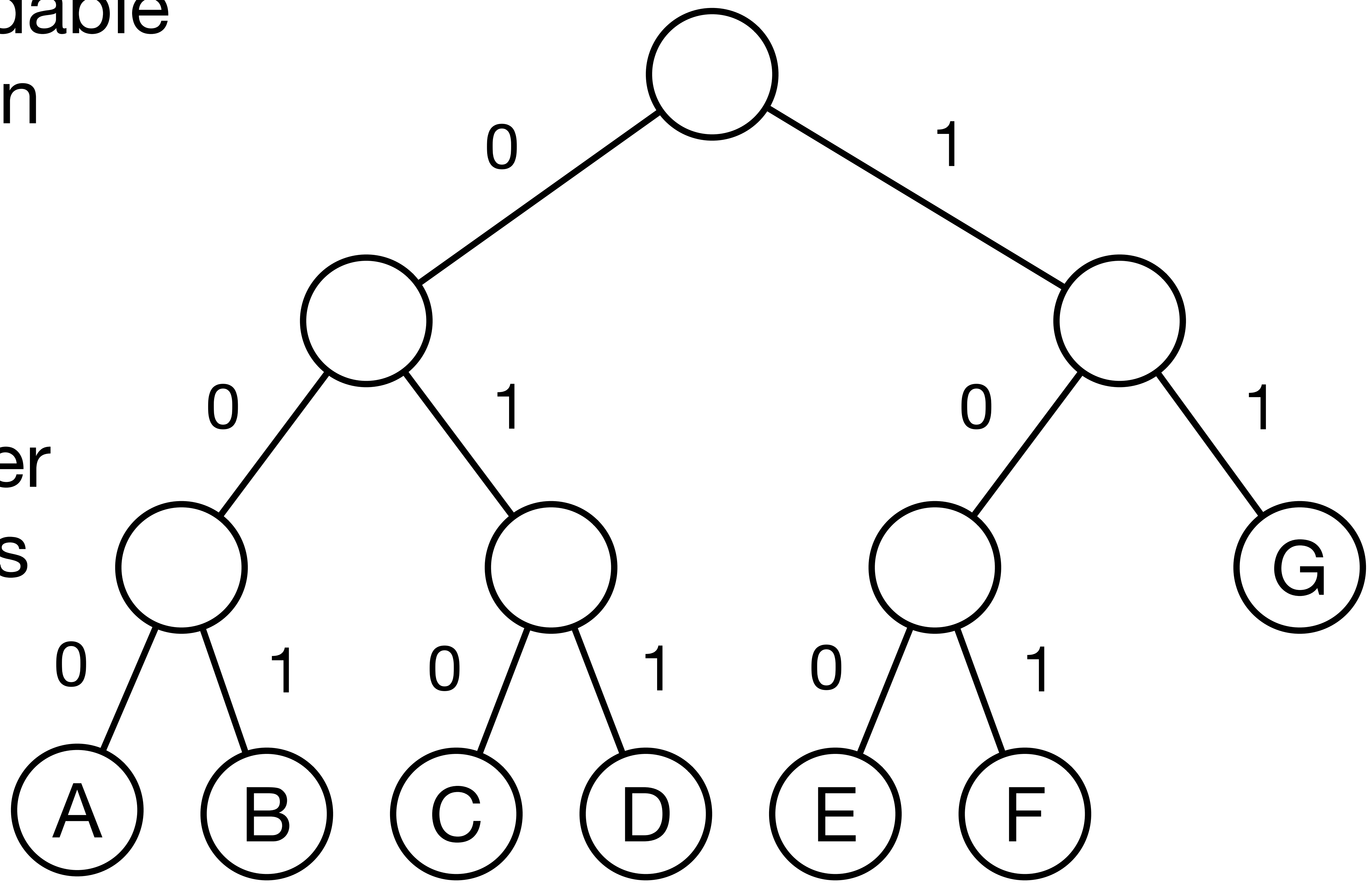
- All sequences of bits are understandable
- All nodes either leaf or has 2 children
  - can promote single child

Prefix code

- No code word is the prefix of another
- Therefore, no chars in internal nodes

Optimal Code

- Minimum cost code (# of bits)



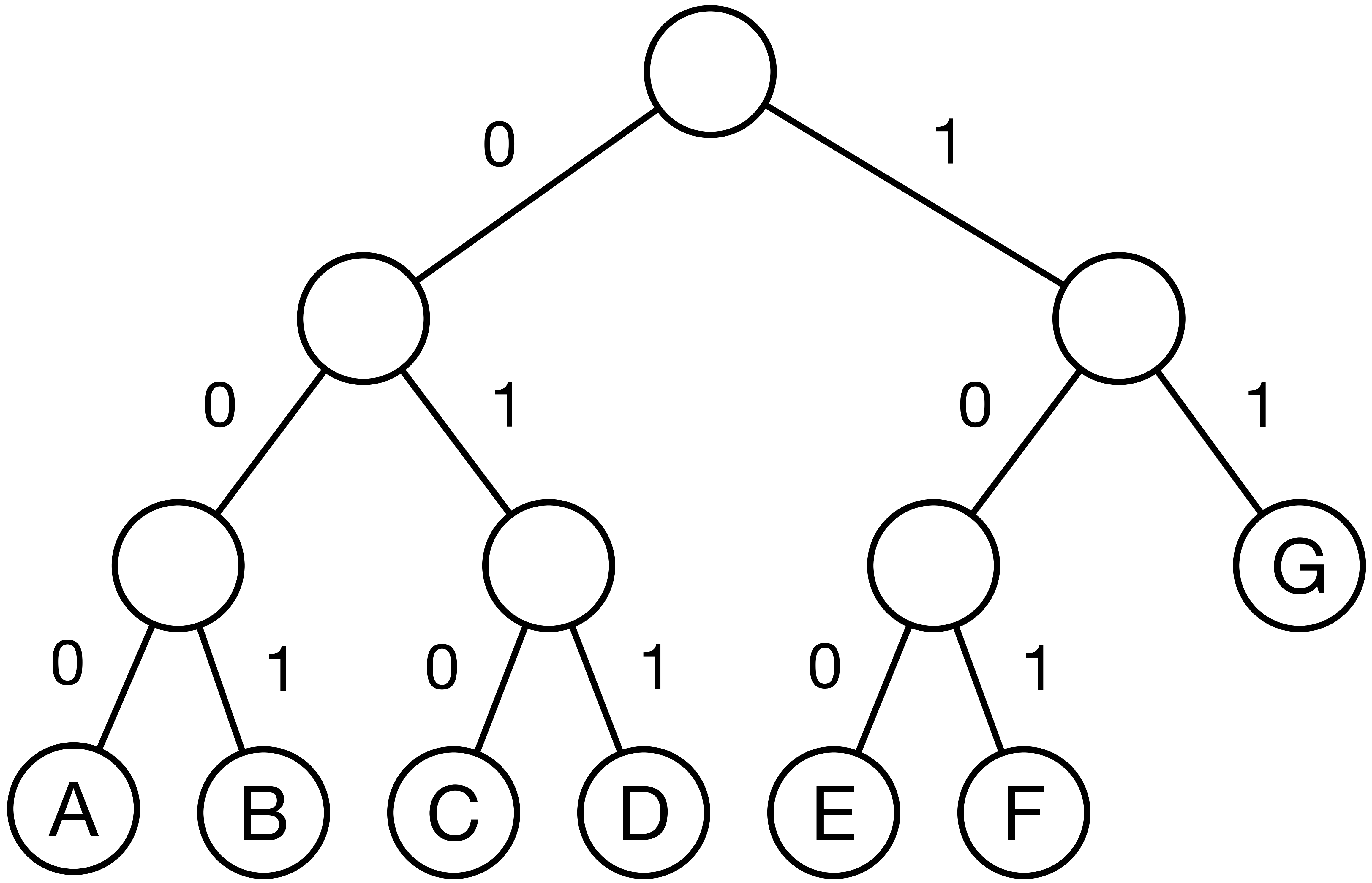
Why do we want the code to be a prefix code? I.e., why do we want it to be the case that no code word is the prefix of another code word?

A. If one code word is the prefix of another, then when decoding, if we see the longer code word, we can't tell if it's the longer one or the shorter one followed by another code word

B. Allowing one code word to be a prefix of another would require longer code words

C. It's easier to represent the code words as a trie if only the leaves have values

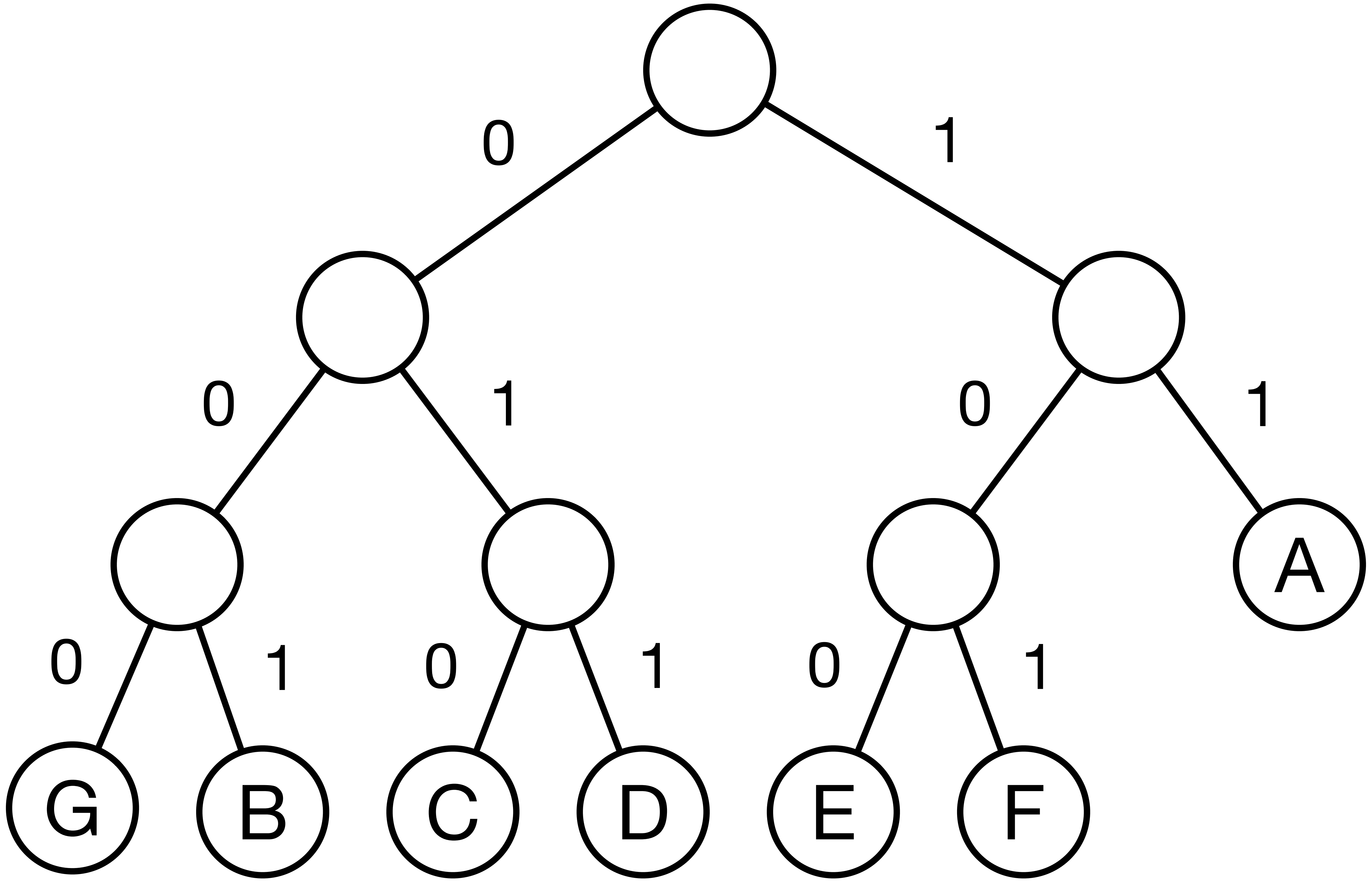
After moving the G up



Letter	Bit rep	Count	Bits used
A	000	13	39
B	001	12	36
C	010	10	30
D	011	5	15
E	100	3	9
F	101	1	3
G	11	1	2

Total length: 39+36+30+15+9+3+2 = 134

Swap the A and G



Letter	Bit rep	Count	Bits used
A	11	13	26
B	001	12	36
C	010	10	30
D	011	5	15
E	100	3	9
F	101	1	3
G	000	1	3

Total length: 26+36+30+15+9+3+3 = 122

How should we select code words for characters?

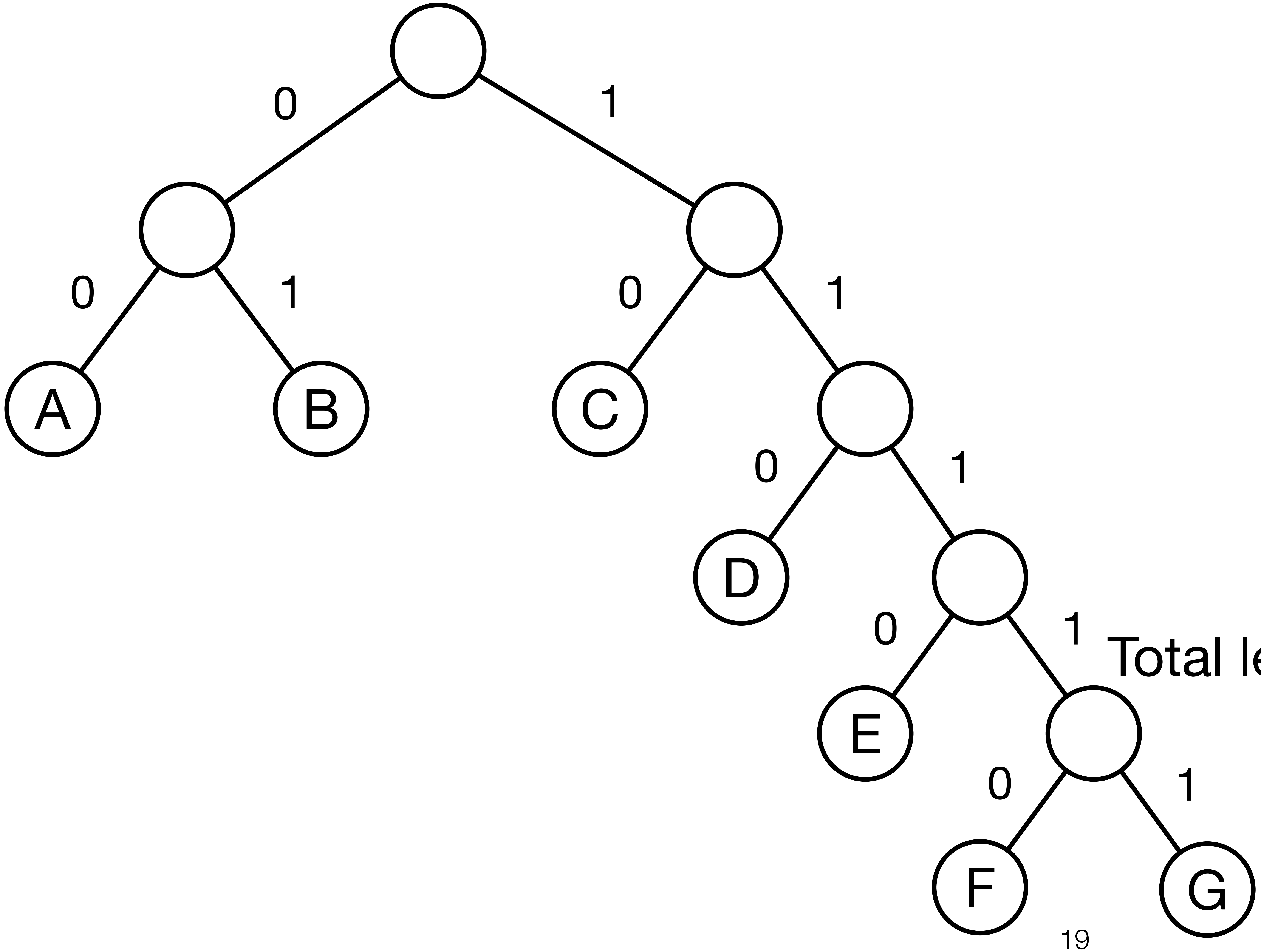
A. The least frequent characters should have the shortest code words

B. The most frequent characters should have the shortest code words

C. All code words should have the same length and be ordered alphabetically

D. It doesn't matter how we assign code words to characters

Optimal length code



Letter	Bit rep	Count	Bits used
A	00	13	26
B	01	12	24
C	10	10	20
D	110	5	15
E	1110	3	12
F	11110	1	5
G	11111	1	5

Total length: 26+24+20+15+12+5+5 = 107

Huffman's Algorithm

Algorithm to create optimal prefix code

David A. Huffman published it in 1952

Idea: keep forest of trees, merge two smallest trees at each step

Huffman's Algorithm

Count the number of times each letter is used

Create list of singleton nodes (trees) per letter with counts as value

While two or more nodes are in the list

- select the two smallest nodes
- make them leaves of a new node whose value is the sum of their counts

Traverse tree to generate strings for all leaf nodes

Example: bad cabbage beef

Counts:

Forest:

Combine:

Example: bad cabbage beef

Counts:

a	b	c	d	e	f	g	space
3	4	1	1	3	1	1	2

Forest:

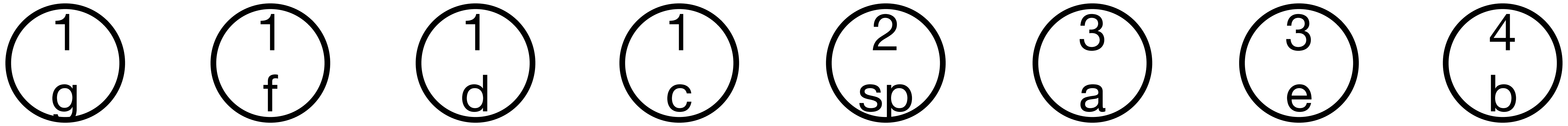
Combine:

Example: bad cabbage beef

Counts:

a	b	c	d	e	f	g	space
3	4	1	1	3	1	1	2

Forest:



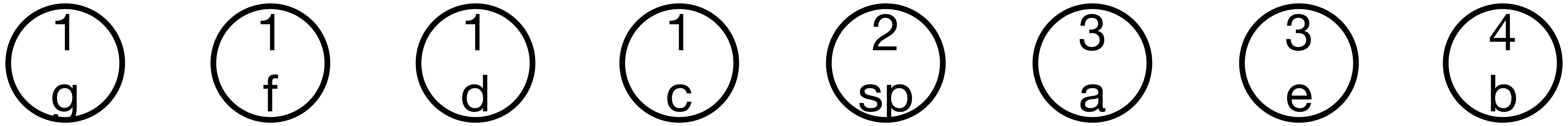
Combine:

Example: bad cabbage beef

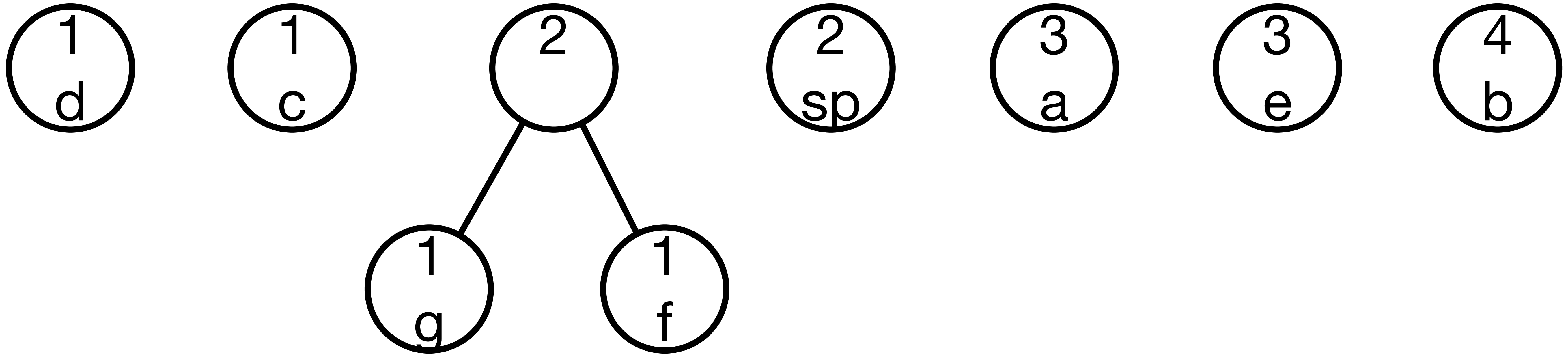
Counts:

a	b	c	d	e	f	g	space
3	4	1	1	3	1	1	2

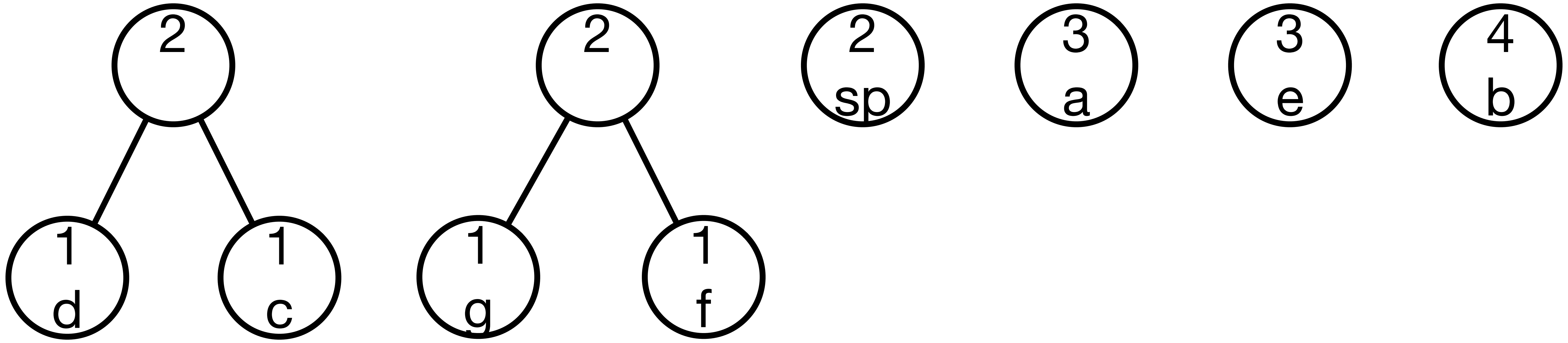
Forest:



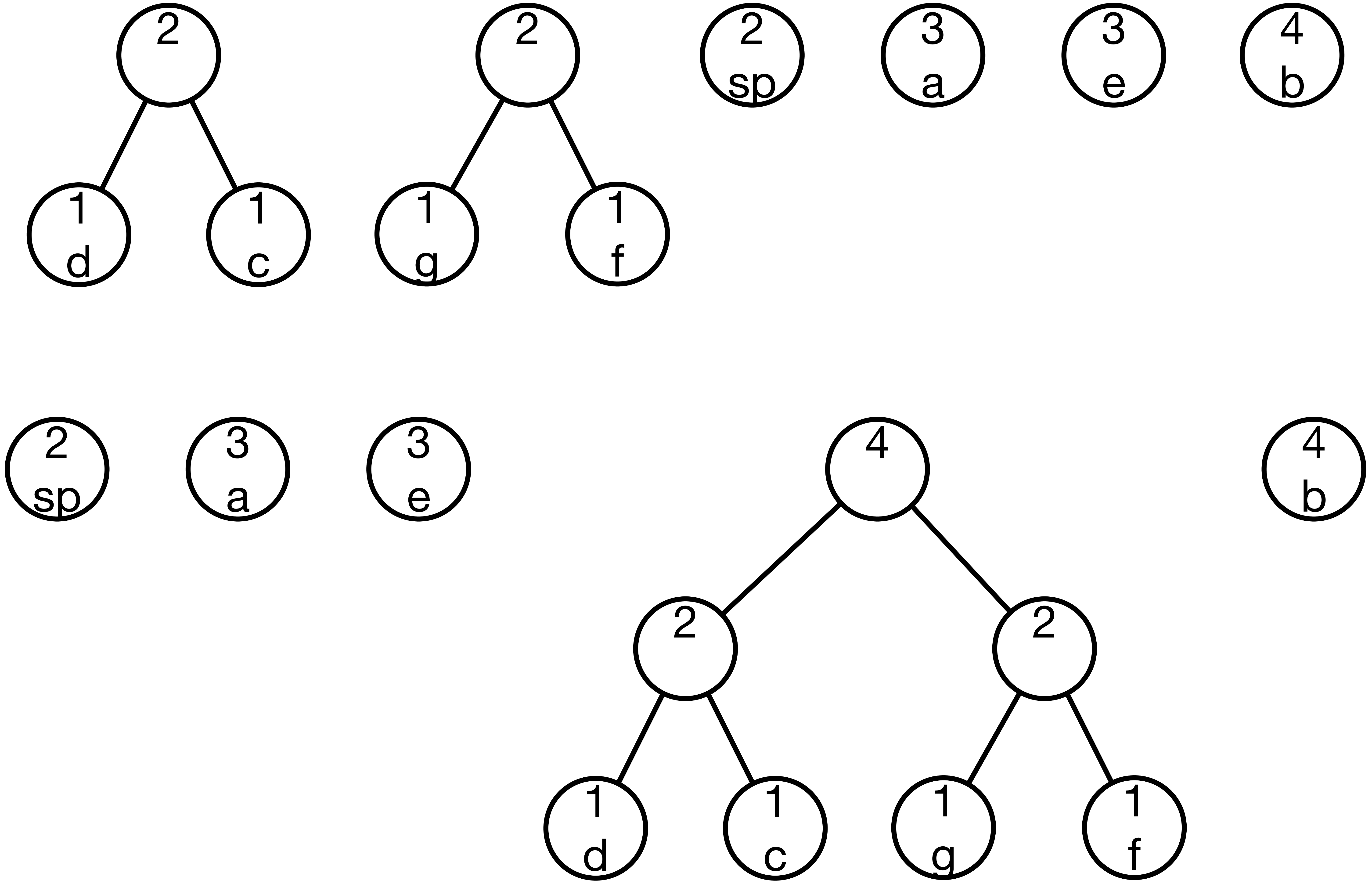
Combine:



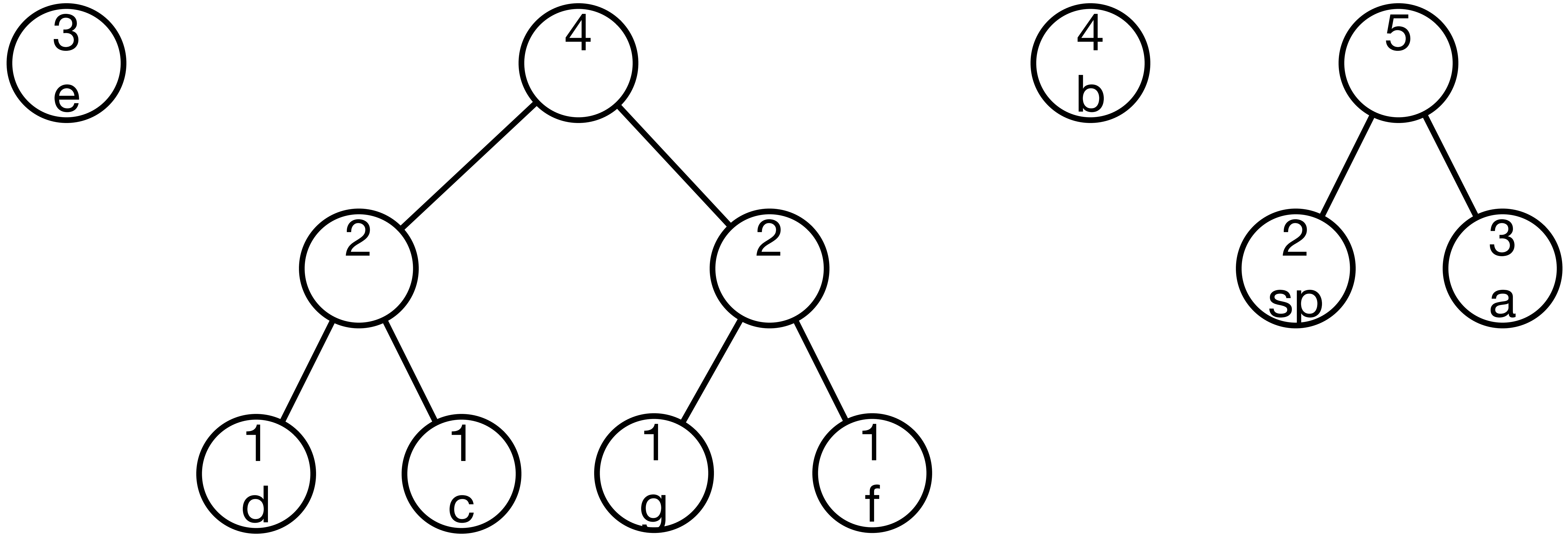
Example continued



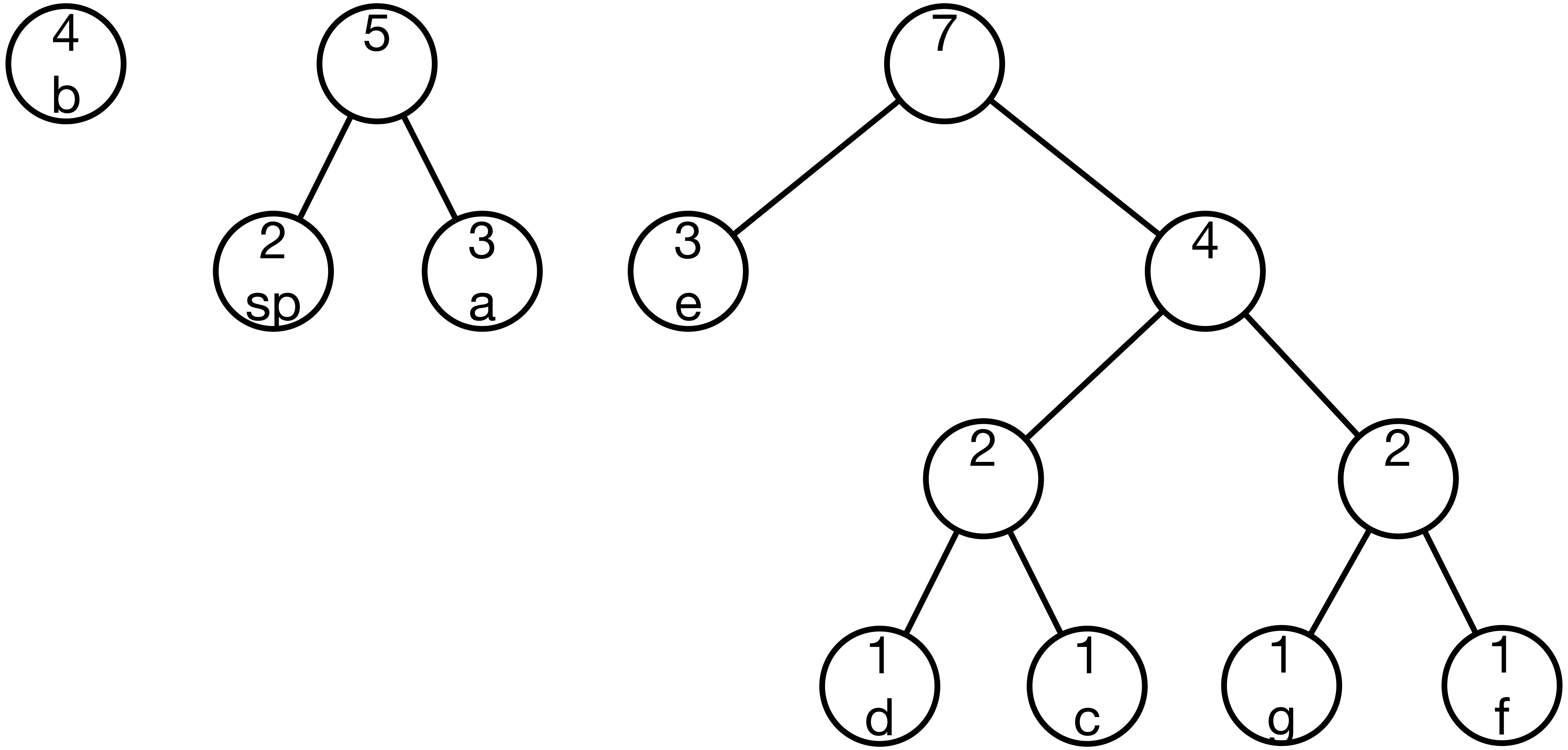
Example continued



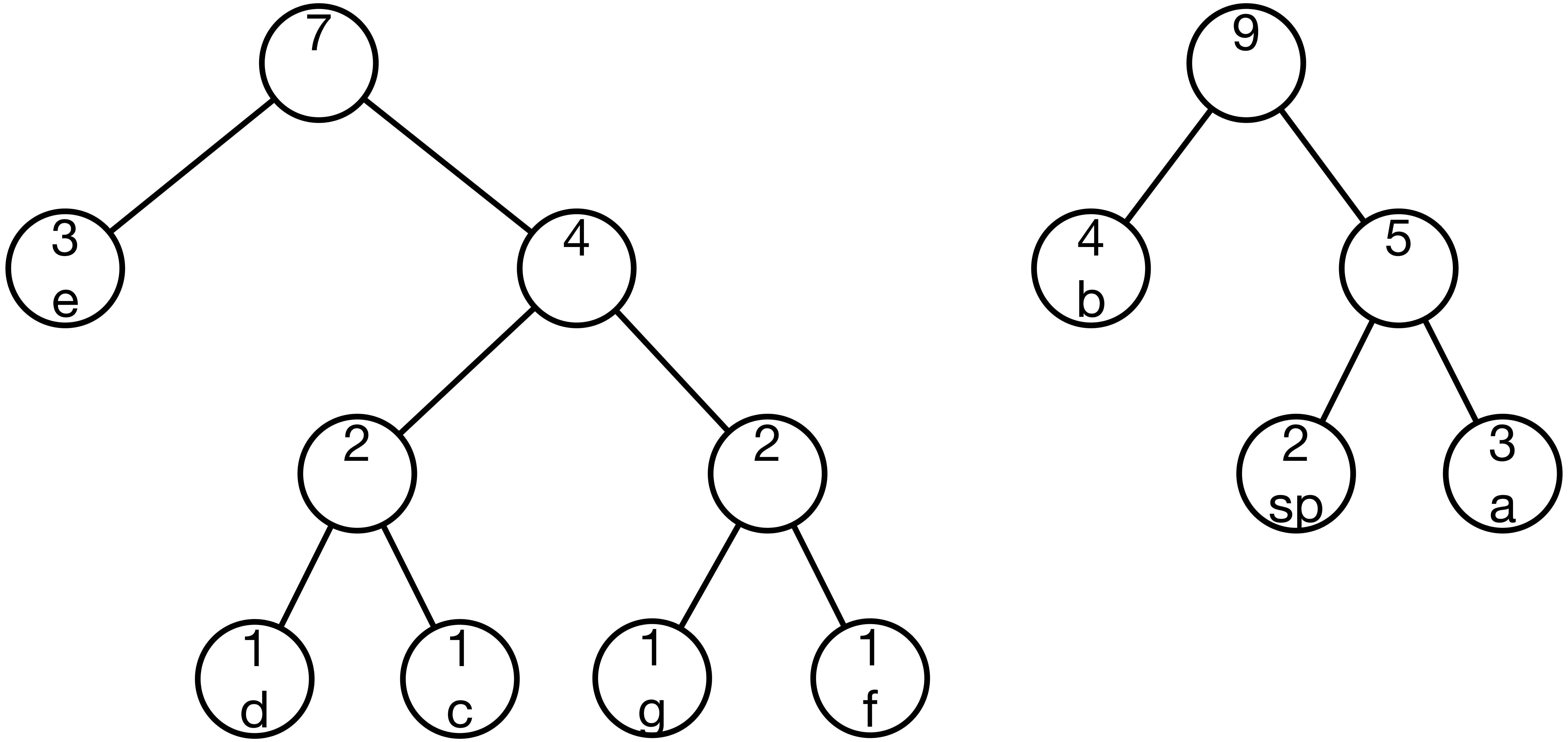
Example continued



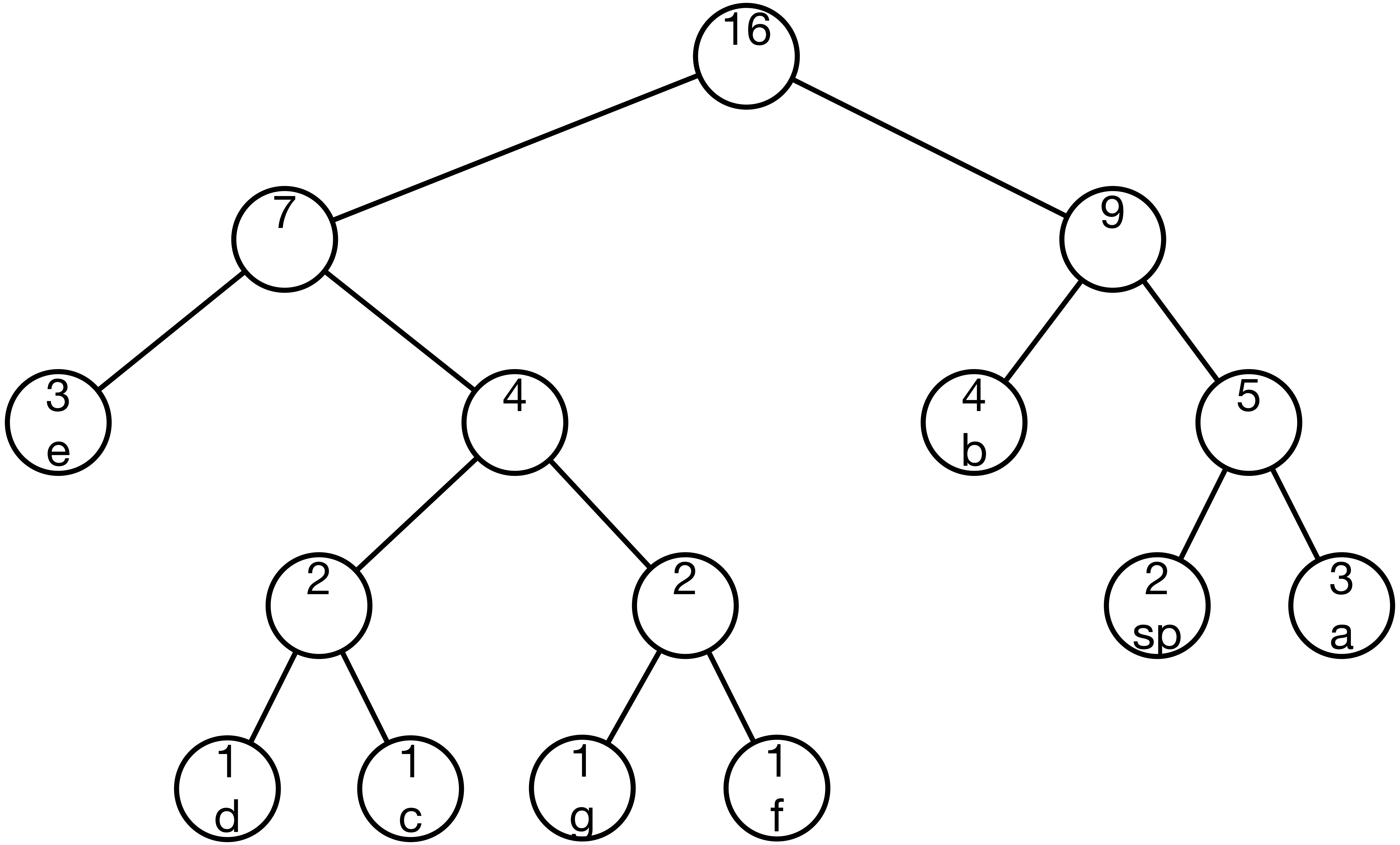
Example continued



Example continued

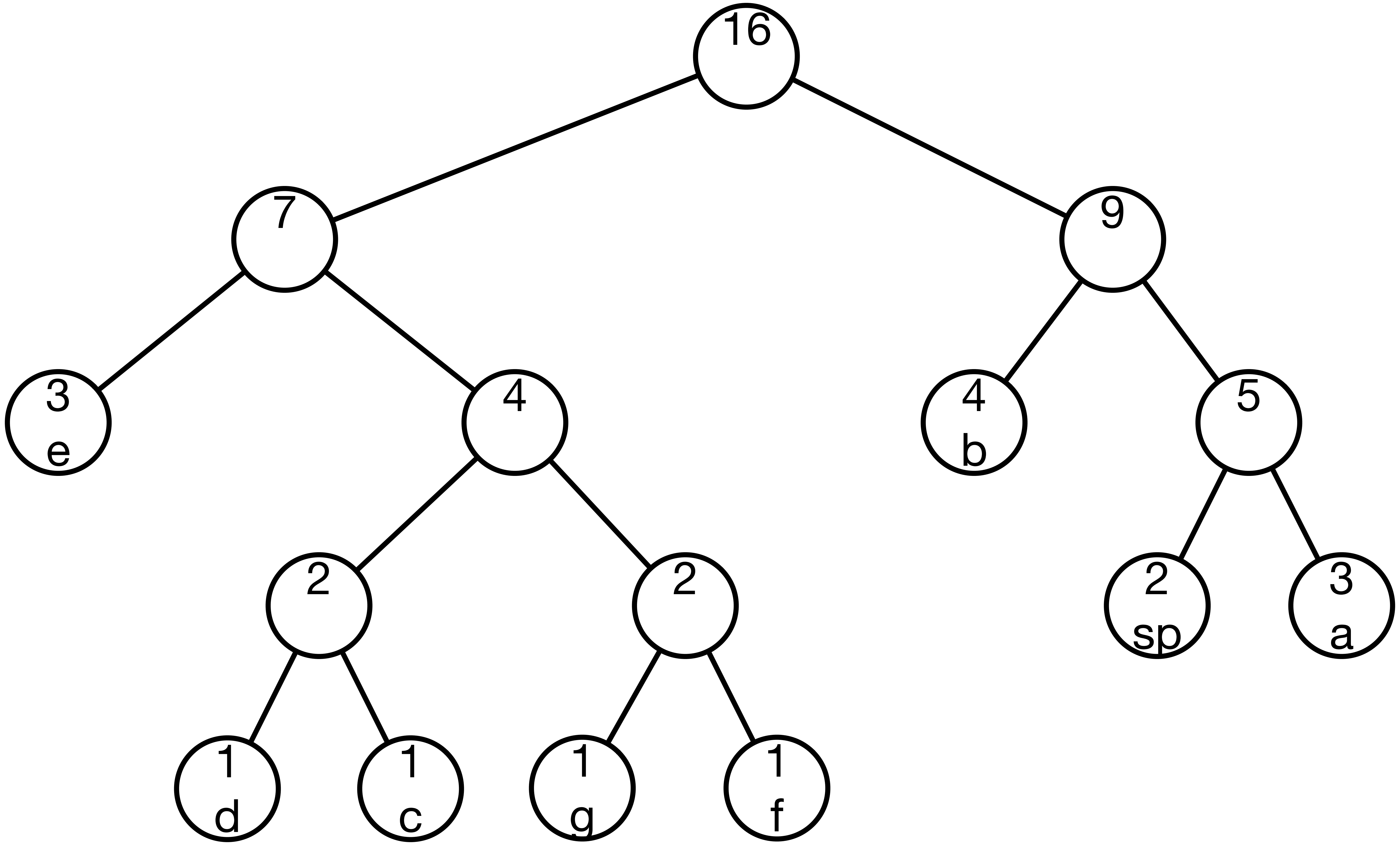


Example continued



Example continued

Letter	Bit rep
a	111
b	10
c	0101
d	0100
e	00
f	0111
g	0110
space	110



In-class exercise

Create a Huffman tree for `oberlin college`