Application of Priority Queues/Heaps:

Huffman Codes

Problem:

How to compress a file without losing information?

Compression saves space and, more importantly, saves time.

Example:

Let m be a message of size 100,000 consisting of the letters ‘a’, ‘b’, ‘c’, ‘d’, ‘e’.

Encode m into e, a sequence of bits.

In the Unicode collating sequence, 16 bits are allocated for each character.

If we use that encoding,

$|E|$, the size of E, is 1,600,000

Suppose we use a fixed number of bits for each character. What is the minimum number of bits needed to encode 5 characters uniquely?

If each character is represented with the same number of bits, 3 bits per character are needed to encode 5 characters uniquely.

In general, the minimum number of bits needed to encode all n characters uniquely is the number of bits in the binary representation of n:

$$\text{ceil}(\log_2 n)$$

ceil(x) returns the smallest integer greater than or equal to x.

Here is a possible encoding:

‘a’ ⟶ 000
‘b’ ⟶ 001
‘c’ ⟶ 010
‘d’ ⟶ 011
‘e’ ⟶ 100

$$|E| = 100000 * 3 = 300000$$

Can we do better? We would like an encoding of *n* characters that does not require ceil(log2*n*) bits per character.

How about this:

‘a’ → 0
‘b’ → 1
‘c’ → 00
‘d’ → 01
‘e’ → 10

$|E| << 300{,}000$

But

There is ambiguity:

001010 could be the encoding of

adda or cee or …

There is ambiguity because some
of the character encodings are prefixes of
other character encodings.

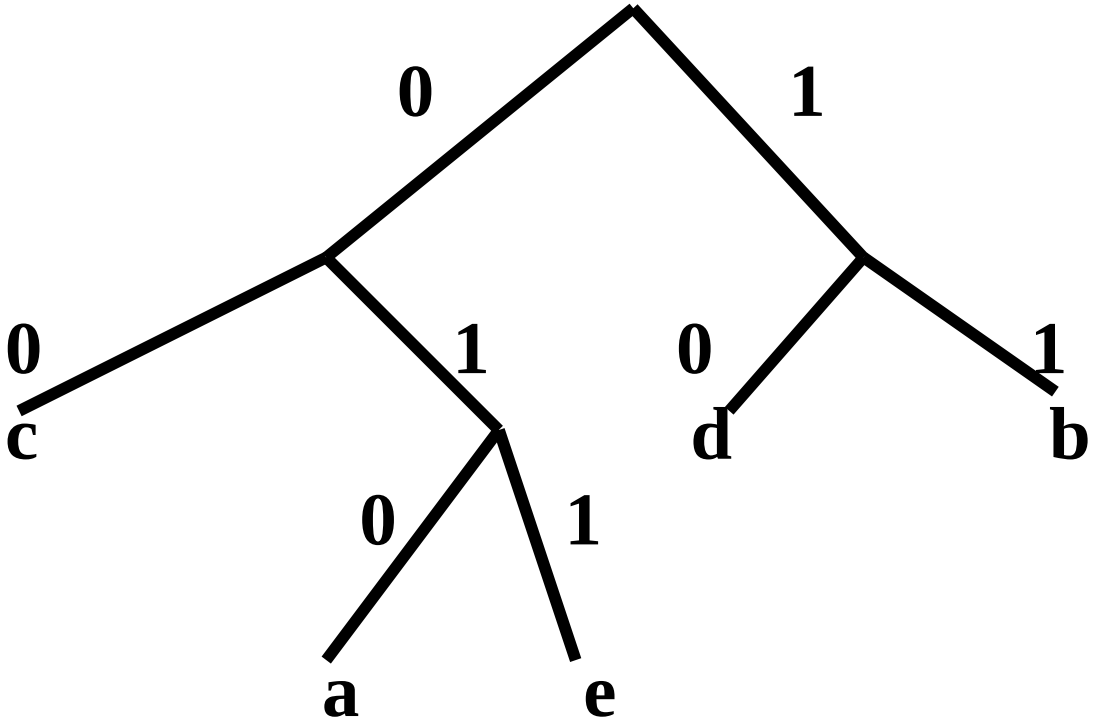
A prefix-free encoding will be unambiguous!

To get a prefix-free encoding, create a binary tree:

Left branch for a 0 bit

Right branch for a 1 bit

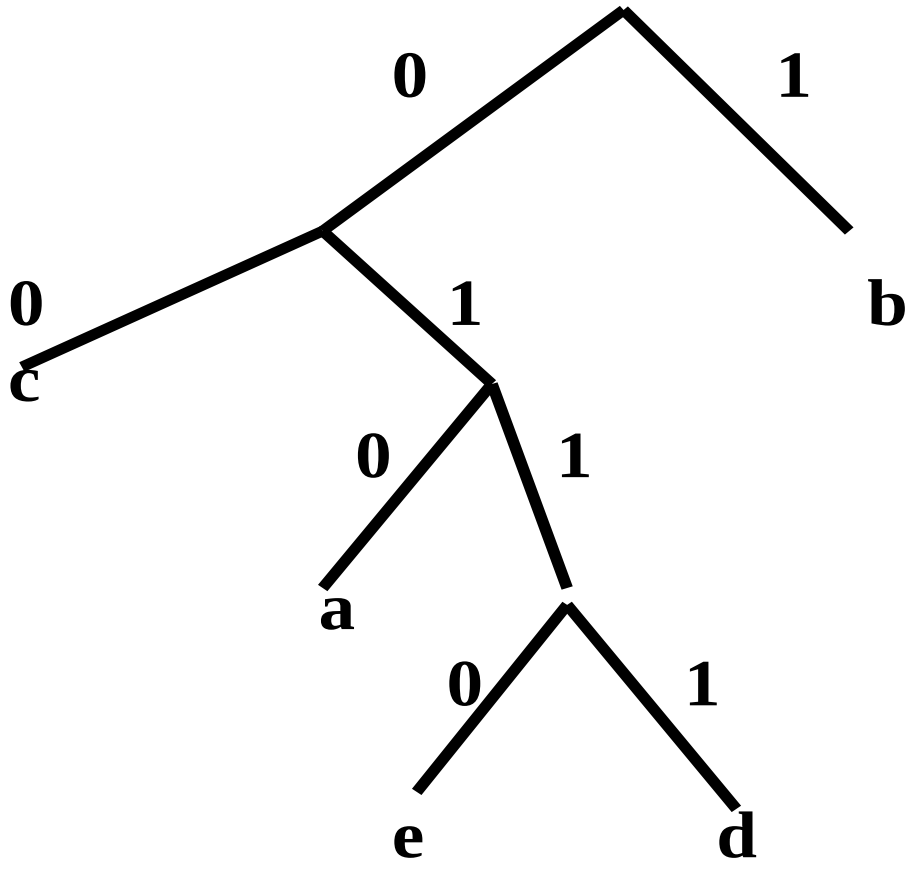
Each character will be a leaf.


0
1
0
1
0
1
c
d
b
0
1
a
e

Since each character is a leaf, no two characters can be on the same path from the root, so this gives a prefix-free – and unambiguous – encoding:

‘a’ ⟶ 010
‘b’ ⟶ 11
‘c’ ⟶ 00
‘d’ ⟶ 10
‘e’ ⟶ 011

Here is another binary tree that produces another unambiguous encoding:



Given an unambiguous encoding into E, to determine |E|, we need to know the frequency of each character in the original message M.

To obtain a minimal prefix-free encoding, create a binary tree, called a *Huffman tree*, that is actually based on the frequencies of the characters in M.

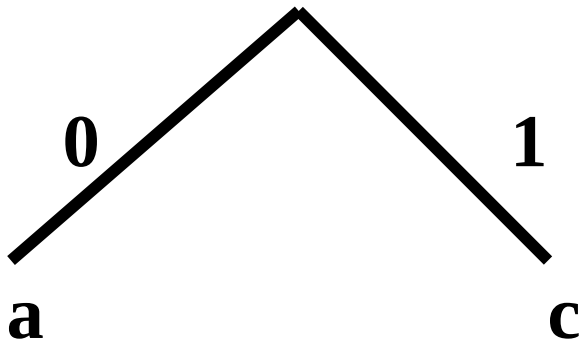
At the level farthest from the root, put the least-frequently occurring characters. Their encodings will have the most bits.

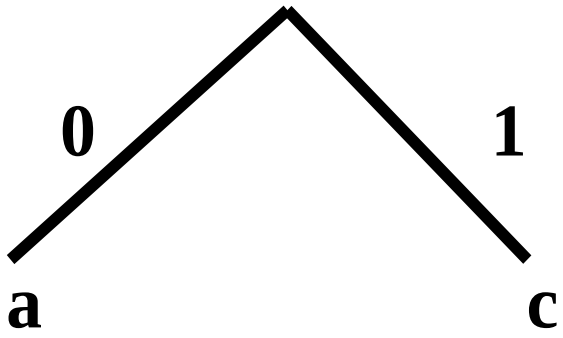
Suppose the frequencies are:

‘a’ 5,000
‘b’ 20,000
‘c’ 8,000
‘d’ 40,000
‘e’ 27,000

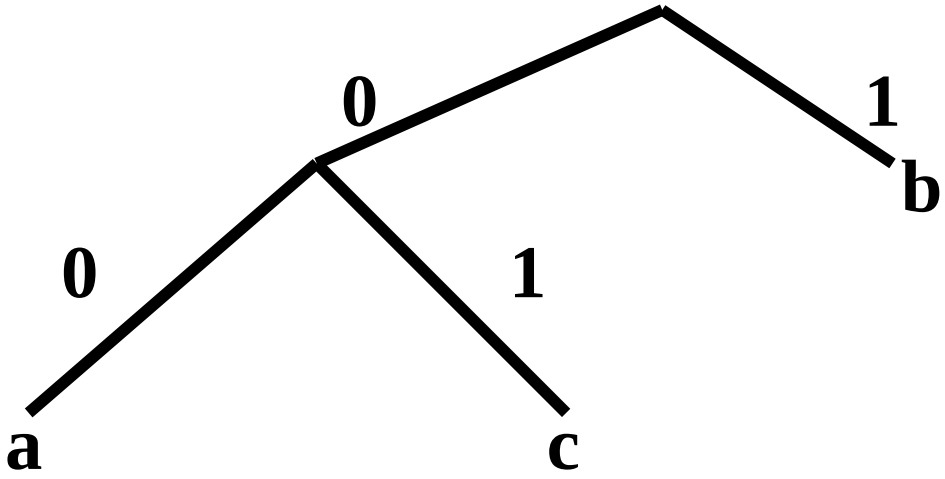
Left-branch for last

Right-branch for next-to-least

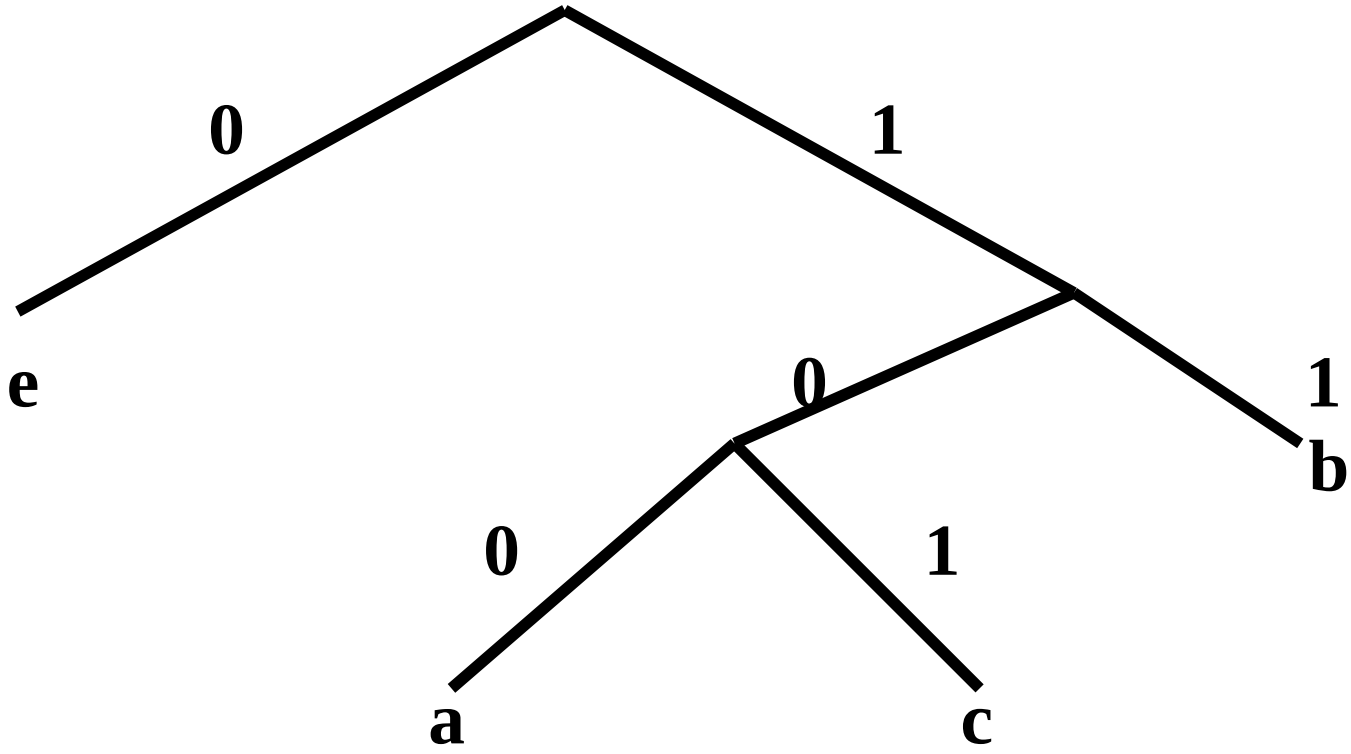

0
1
a
c



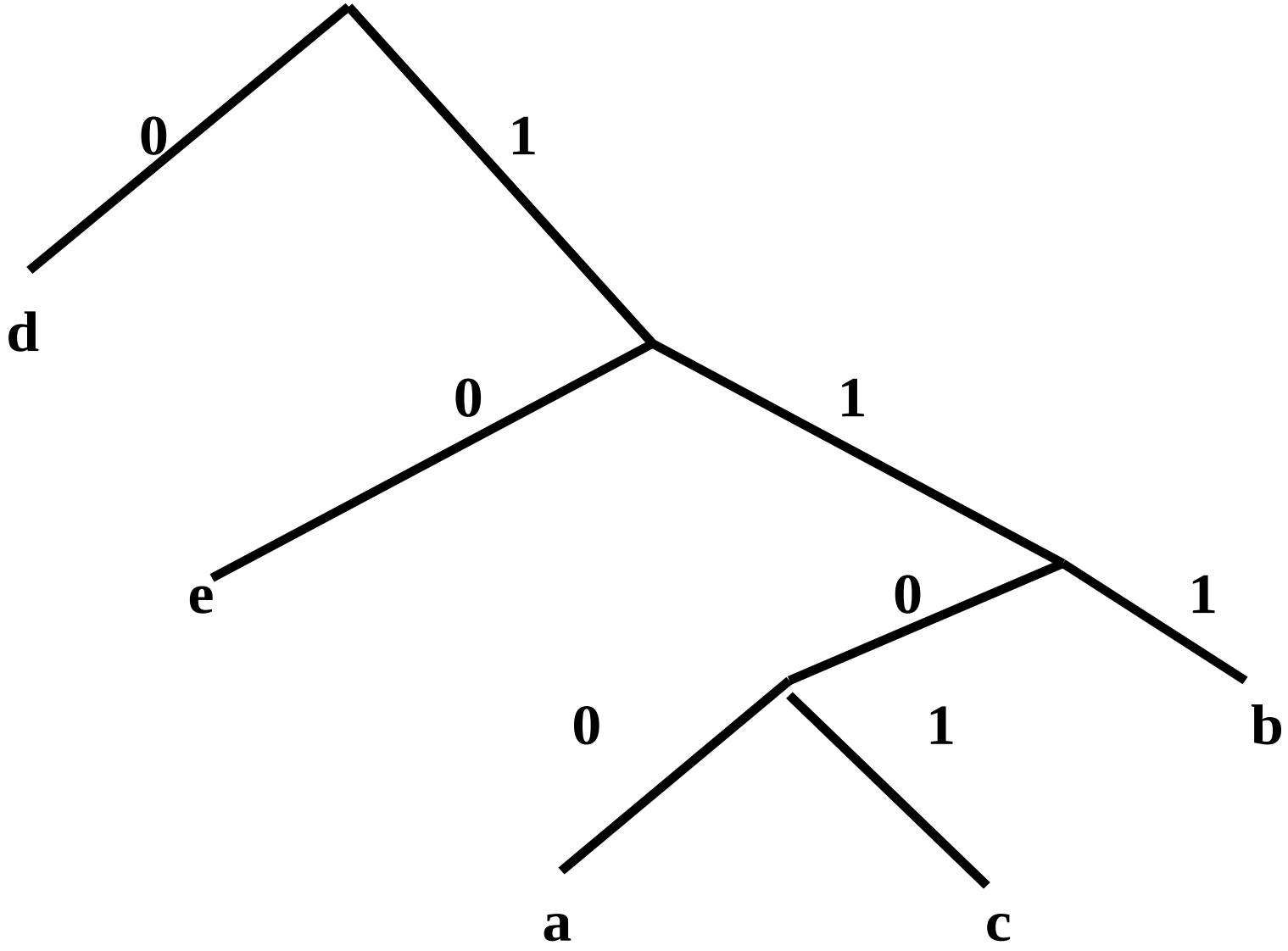
Now what? The sum of these frequencies is 13,000. That sum is less than b's frequency of 20,000, so we'll create 2 more branches: This subtree will be at the end of the left branch, and b will be the leaf of the right branch:


0
1
b
0
1
a
c

The sum of the frequencies of this subtree is 33000, which is greater than 27000, so this subtree should be a right branch, with ‘e’ – frequency of 27000 – as the leaf of the left branch.

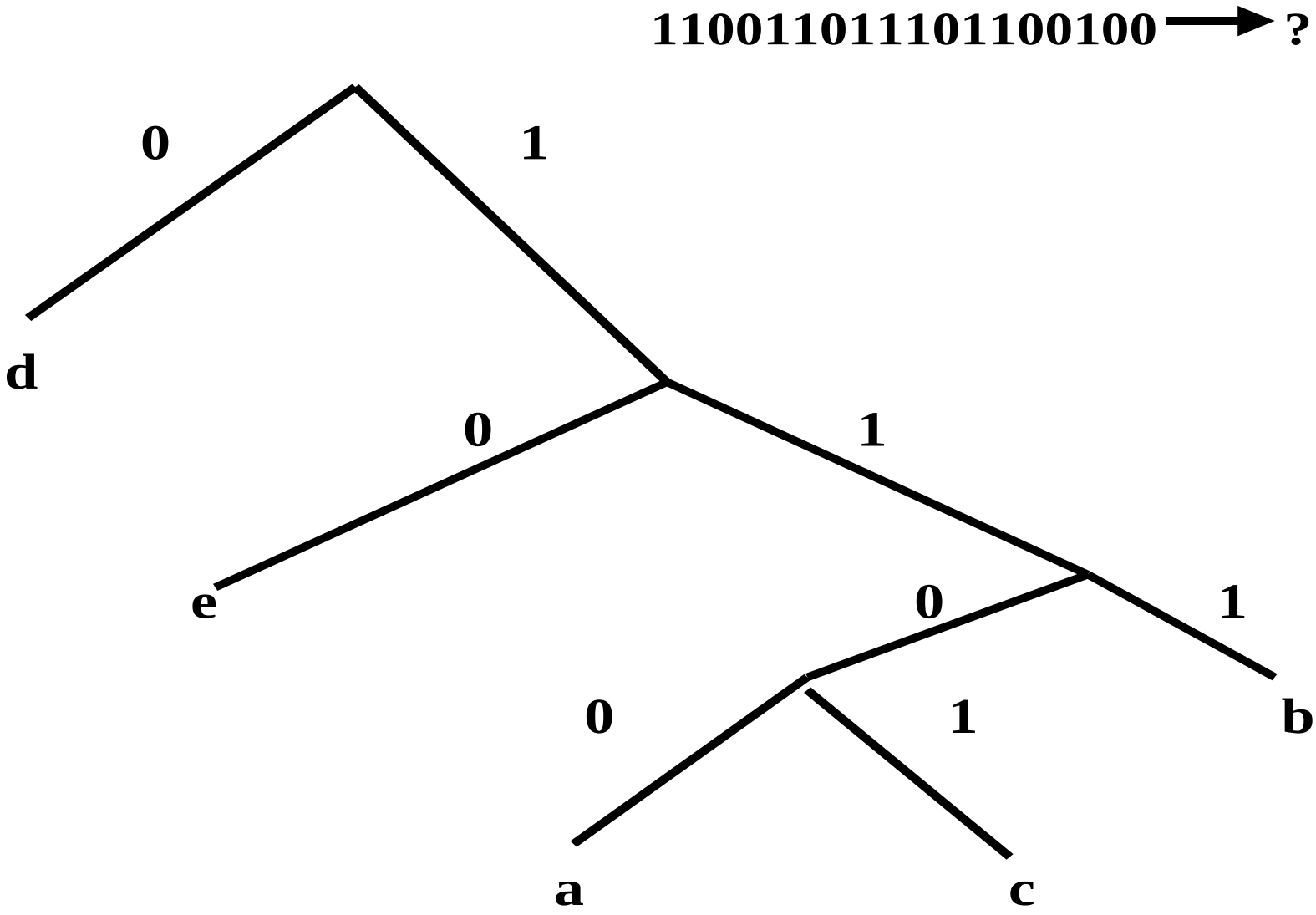

0
1
e
0
1
b
0
1
a
c

Finally, the sum of the frequencies of this subtree is 60000, which is greater than 40000, so this subtree should be a right branch, and ‘d’ – with a frequency of 40000 – should be the leaf of the left branch.


0
1
d
0
1
e
0
1
0
1
b
a
c

‘a’ → 1100
‘b’ → 111
‘c’ → 1101
‘d’ → 0
‘e’ → 10

“acceded” → 11001101111011100100


11001101111011001 00 → ?
0
1
d
0
1
e
0
1
0
1
b
a
c

|E| = the sum, over all characters, of the number of bits for the character times the frequency of the character.

$$
\begin{aligned}
|E| &= 4 * 5000 + \\
&\quad 3 * 20000 + \\
&\quad 4 * 8000 + \\
&\quad 1 * 40000 + \\
&\quad 2 * 27000 \\
\\
&= 206{,}000
\end{aligned}
$$

We will use a priority queue to hold the frequencies of characters and subtrees.

add: Insert the character's frequency into the priority queue

removeMin: Delete the element with *lowest* frequency from the priority queue.

Each element in the priority queue consists of a character-frequency pair (plus left, right, parent,...).

For example, suppose the frequencies are:

(a: 34000) (b: 20000) (c: 31000) (d: 10000) (e: 5000)

The order of the elements in the priority queue is:

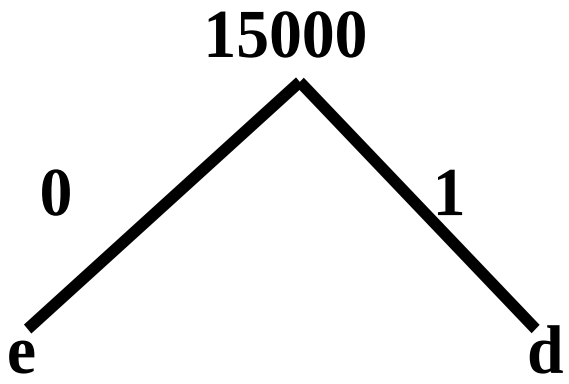
e 5000
d 10000
b 20000
c 31000
a 34000

All we really know is that getMin( ) returns the highest-priority (that is, lowest-frequency) element.

removeMin is called twice. The first element removed becomes a left branch, the second element removed becomes a right branch, and the sum of those frequencies is added to the priority queue:

(: 15000) (b: 20000) (c: 31000) (a: 34000)

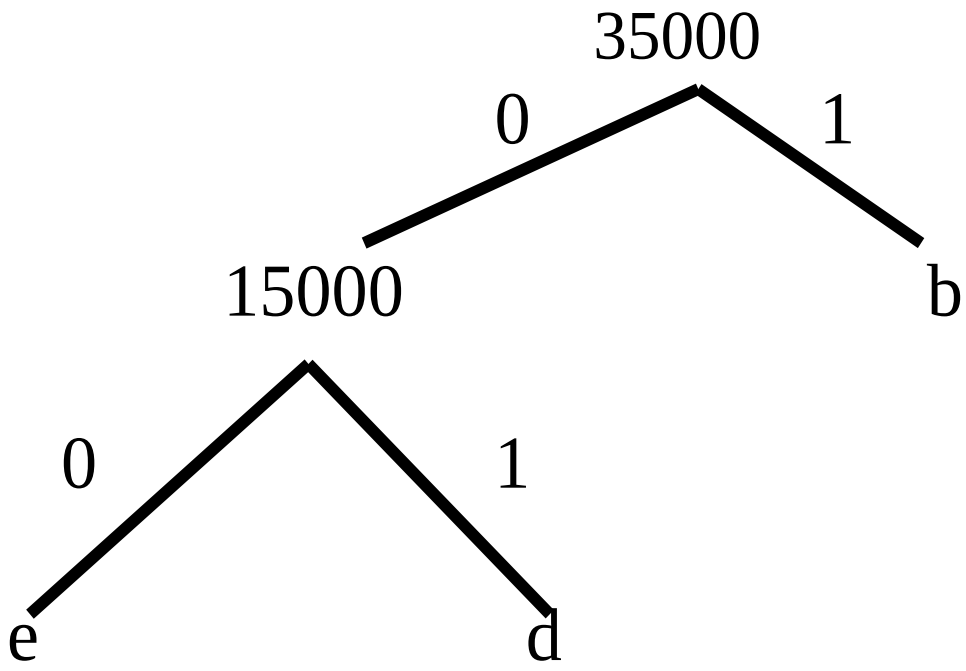
The Huffman tree is now:



Again, removeMin is called twice, and the sum of those frequencies is added to the priority queue:

(c: 31000) (a: 34000) (: 35000)

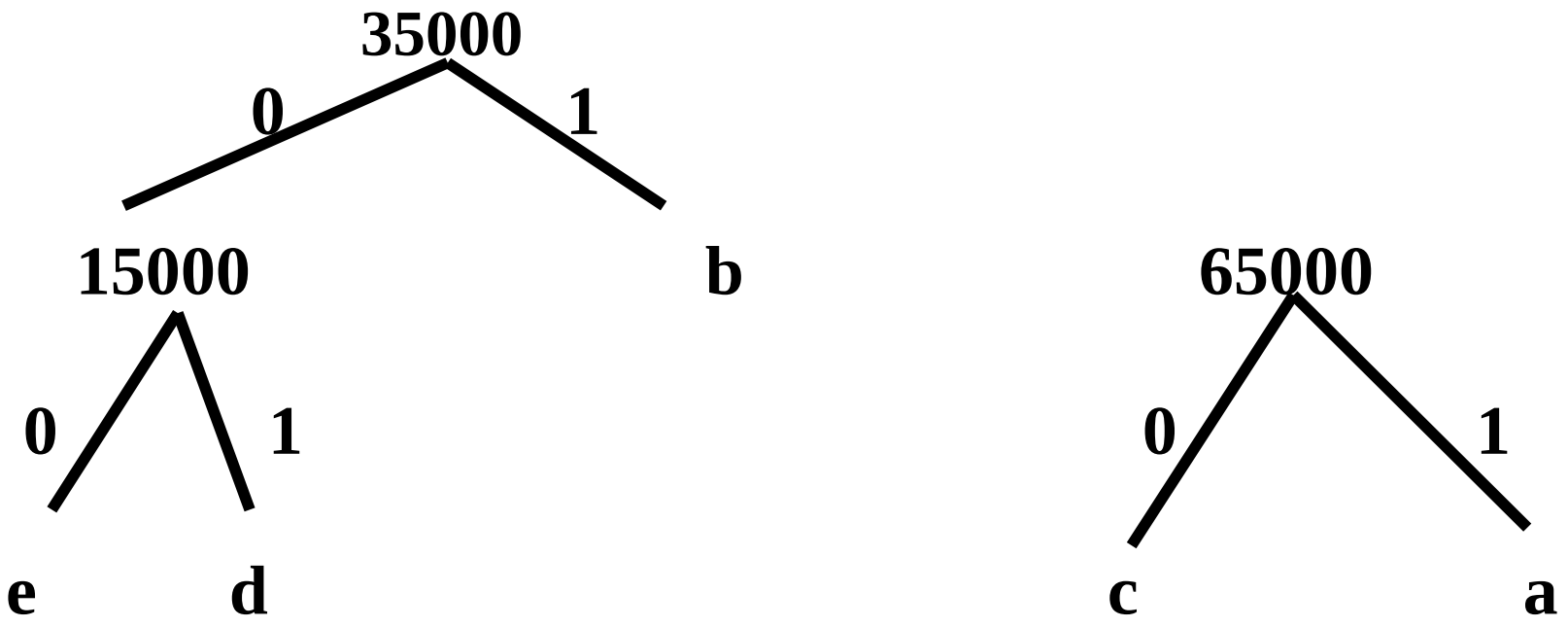
The Huffman tree is now



Again, removeMin is called twice, the elements become leaves in a new Huffman tree, and the sum of those frequencies is added to the priority queue:

(: 35000) (: 65000)

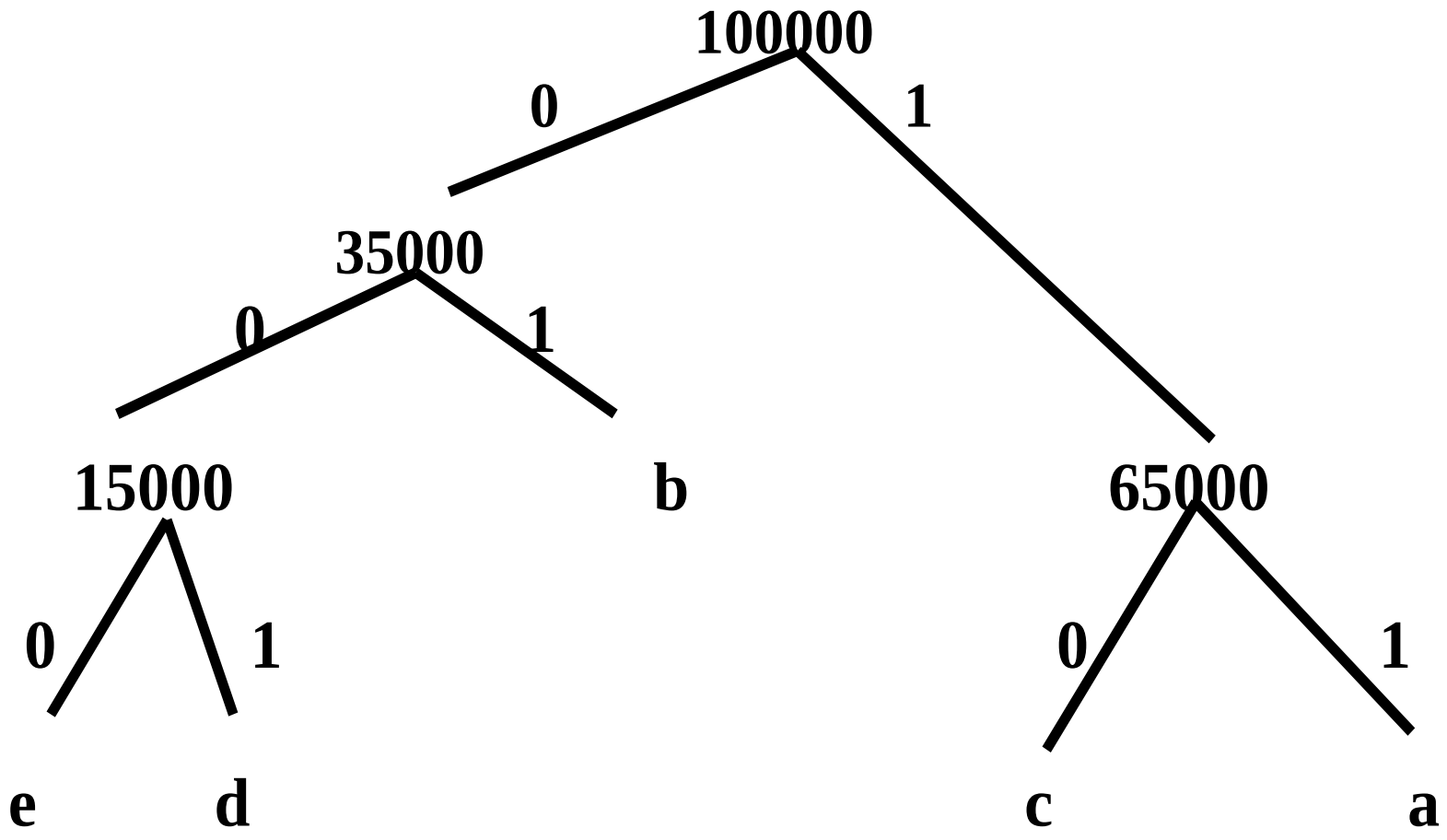
The Huffman trees are now



Finally when (: 35000) and (:65000) are removed from the priority queue and their sum is inserted in the priority queue, the priority queue is:

(: 100000).

The final Huffman tree is:


100000
0
1
35000
0
1
15000
b
65000
0
1
0
1
e
d
c
a

Exercise: Create a minimal, prefix-free encoding for the following character-frequency pairs:

‘a’ 20,000
‘b’ 4,000
‘c’ 1,000
‘d’ 17,000
‘e’ 25,000
‘f’ 2,000
‘g’ 3,000
‘h’ 28,000

Note: You will need to maintain the priority queue, ordered by frequencies.

A *greedy algorithm* is one in which locally optimal choices are made. Sometimes, as with Huffman trees, greed succeeds: Locally optimal choices lead to a globally optimal solution.

Specifically, by repeatedly removing the 2 lowest-frequency elements from the priority queue, and then creating a subtree of the Huffman tree, we end up with a minimal (prefix-free) encoding.