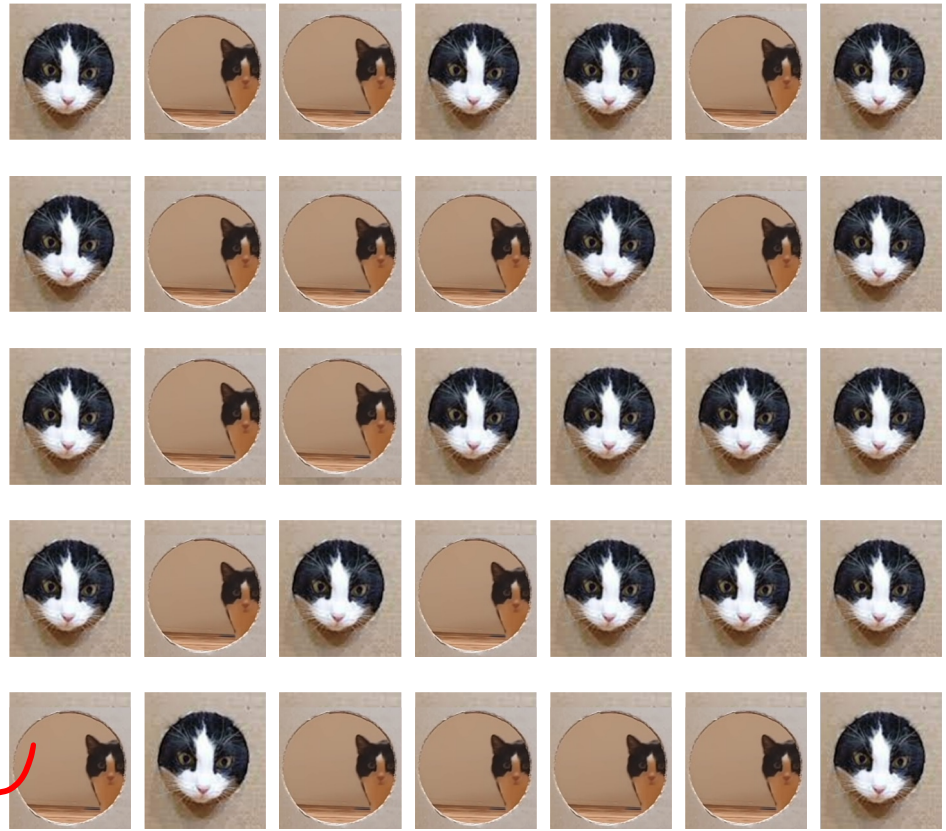
Lecture 19: Huffman Codes and Priority Queues

8/08/22

berryr5@uw.edu



https://www.youtube.com/watch?v=5Cdv932eoU0

A8: Huffman

*Note: This assignment is worth a total of 30 points. It is divided into two parts, each worth approximately half of the points. Please note that this assignment **cannot** be resubmitted and solutions to this homework will not be accepted after 11:59 pm on Friday, August 19th.*

This assignment will assess your mastery of the following objectives:

- Implement a well-designed Java class to meet a given specification.
- Implement, manipulate, and traverse a binary tree.
- Implement the `Comparable` interface
- Follow prescribed conventions for code quality, documentation, and readability.

Cats and Codes

How many patterns can you make with 3 cats?

2x2x2 8

3

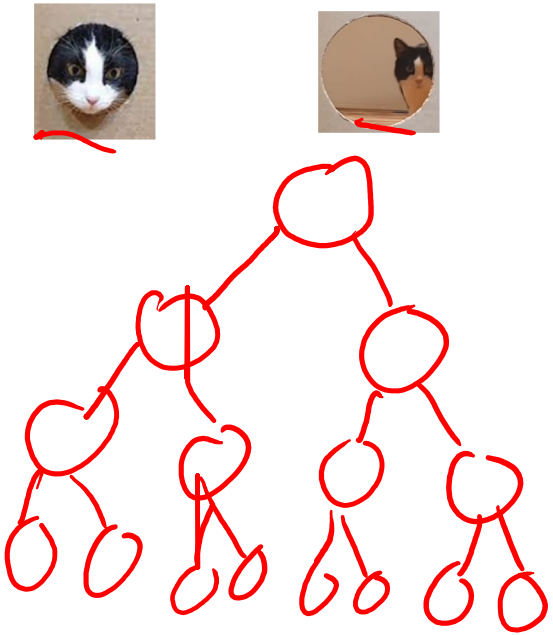
3

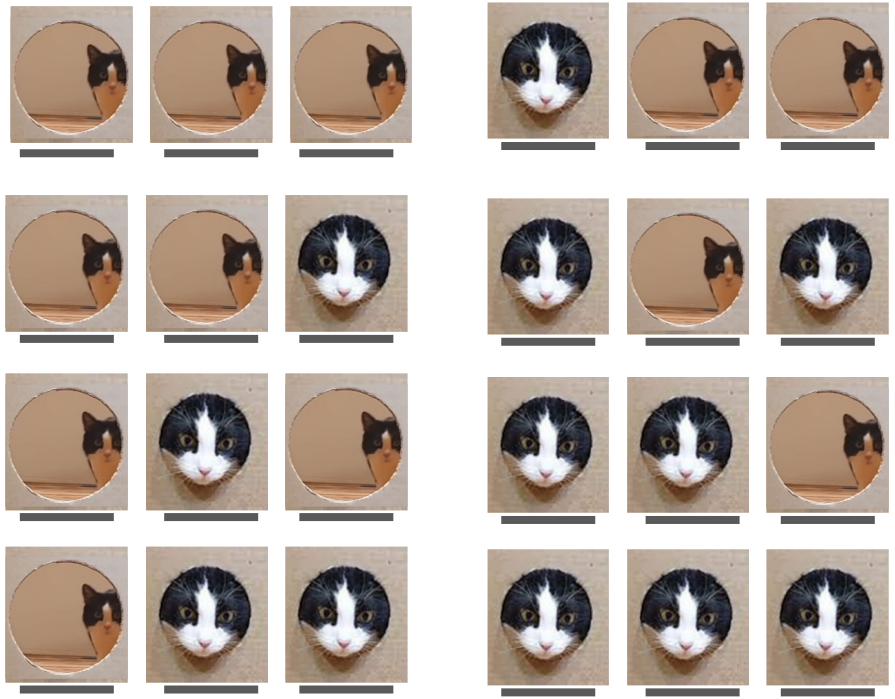
_ _ _

Cats and Codes

How many patterns can you make with 3 cats?

2^3



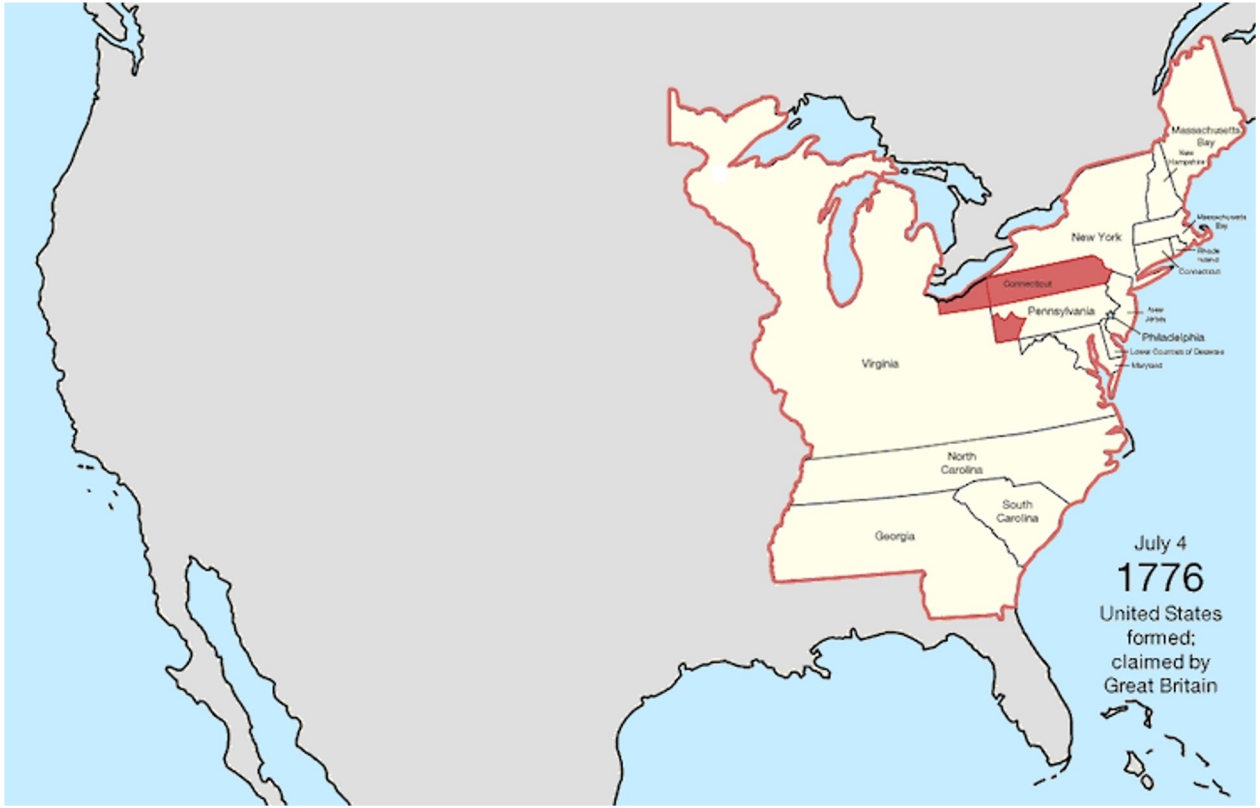


Cats and Codes

How many encoding can you make with 3 bits?

2^3

Herman Hollerith Story - Census



Hollerith Story - Punched Card

A pantograph used to create punch cards

IBM

Hollerith's electronic tabulator, 1902.

More Info: https://www.census.gov/history/www/innovations/technology/the_hollerith_tabulator.html

Ascii

American

Standard

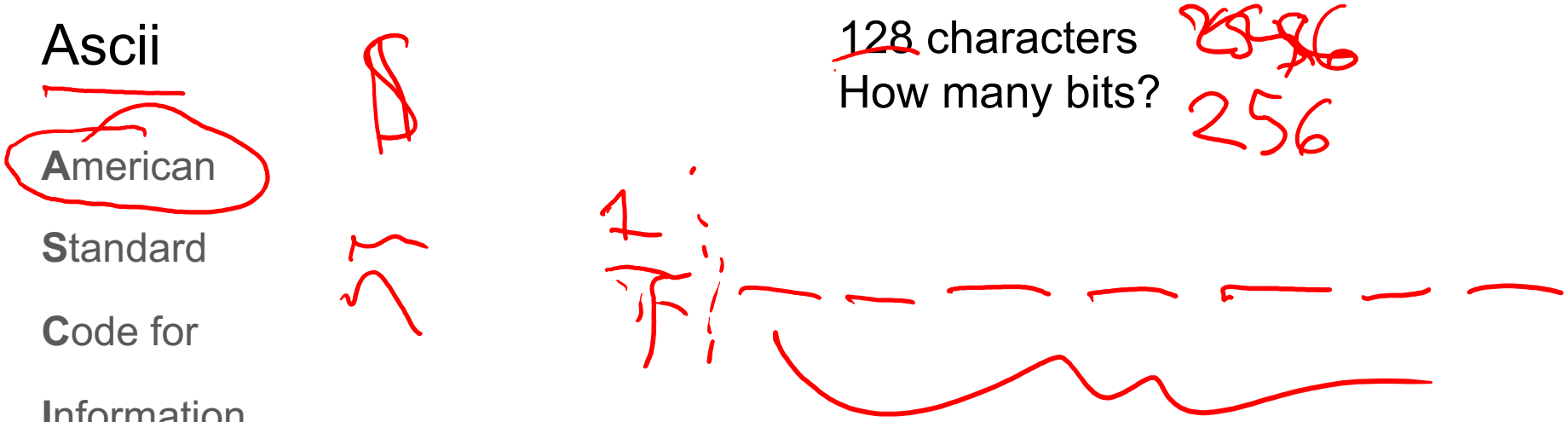
Code for

Information

Interchange

128 characters

How many bits?



Ascii

American

Standard

Code for

Information

Interchange

128 characters
How many bits?

7

Unicode (UTF-8)

What can we do to compress some text?

- Diffent code sizes for diffent characters
- Diffent frequency
  - Code for specific words
  - encode symbols

Making a HuffmanCode - Example

Text:

babyyaxcab

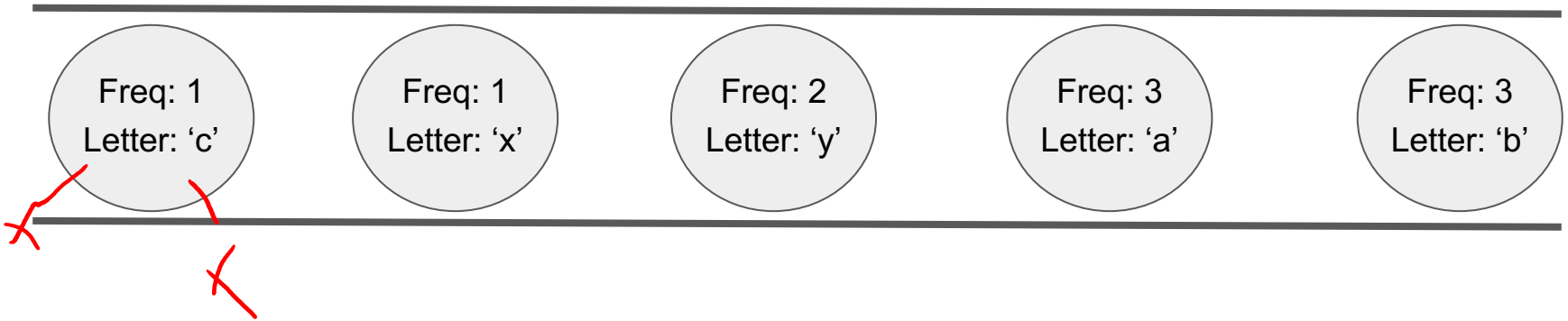
Extracted Letters:

c x yy aaa bbb

Making a HuffmanCode

Extracted Letters:

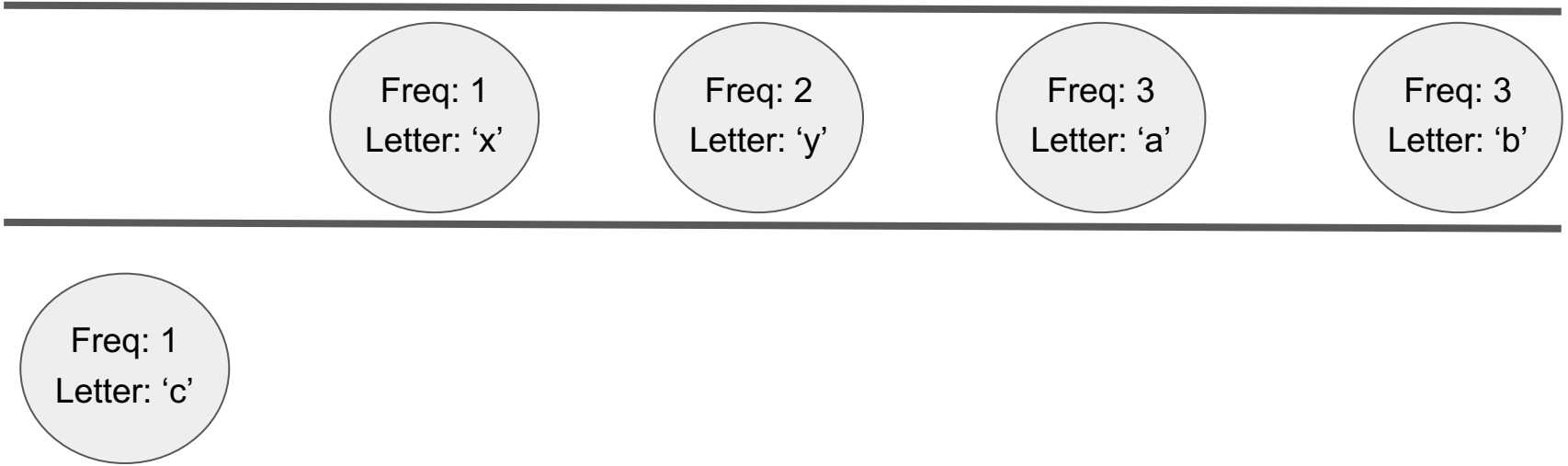
c x yy aaa bbb



Making a HuffmanCode

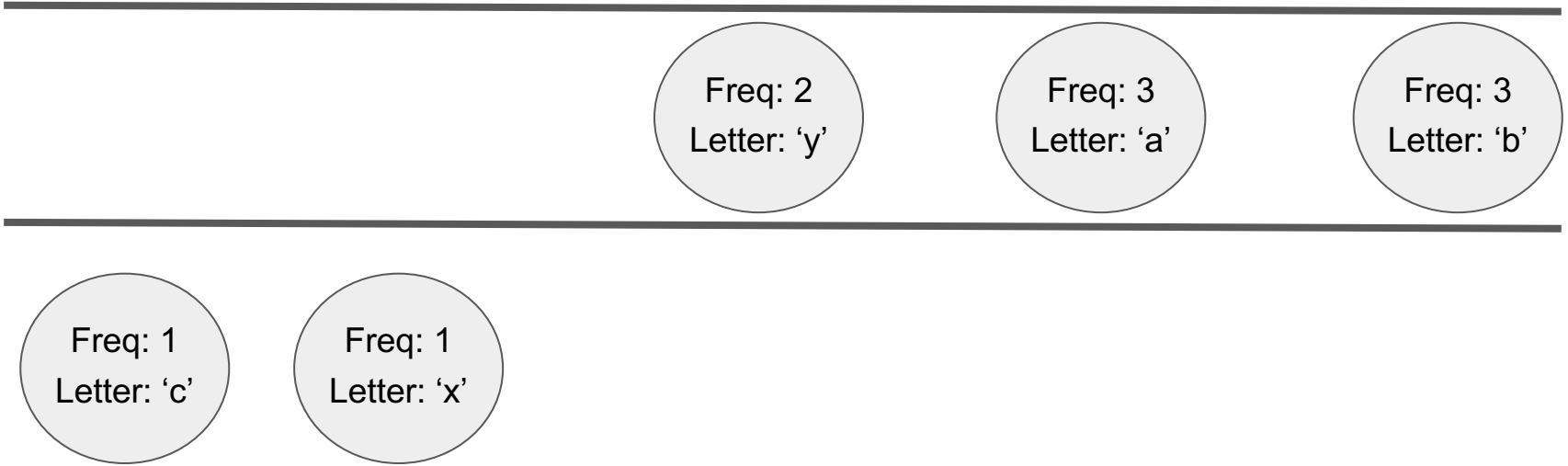
Extracted Letters:

c x yy aaa bbb



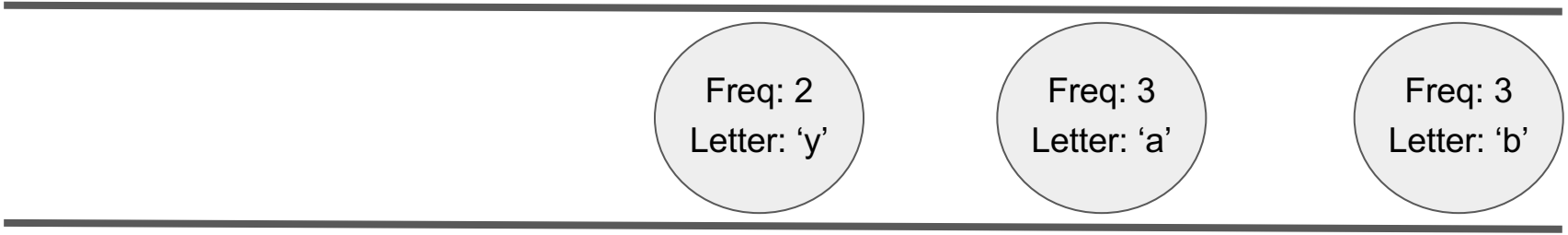
Making a HuffmanCode

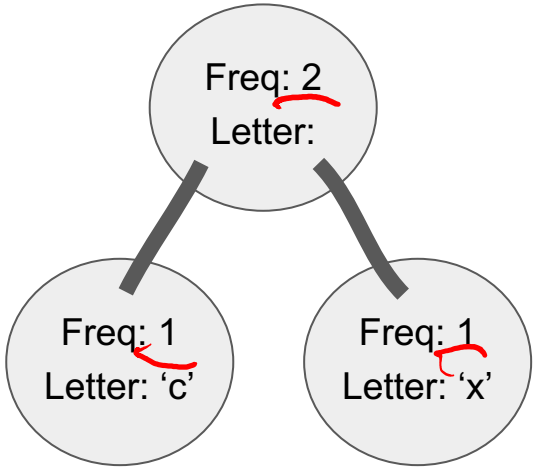
Extracted Letters:
c x yy aaa bbb



Making a HuffmanCode

Extracted Letters:
c x yy aaa bbb

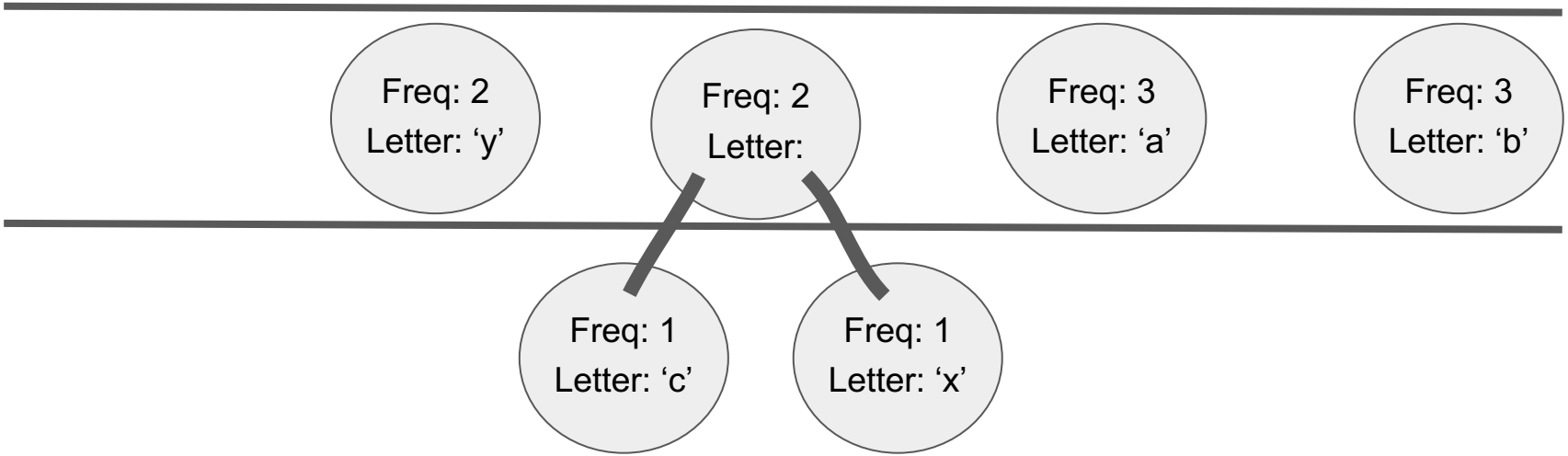




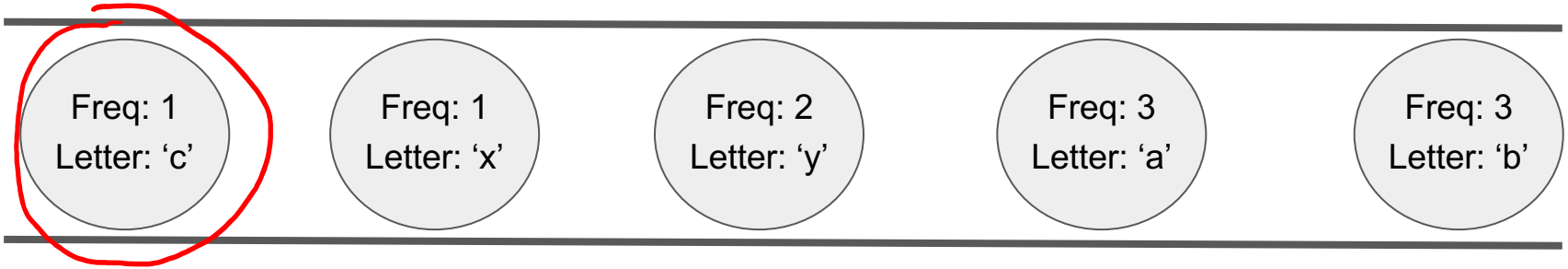
Making a HuffmanCode

Extracted Letters:

c x yy aaa bbb



Making a HuffmanCode - Priority Queue



Priority Queue

A collection of ordered elements that provides fast access to the minimum (or maximum) element.

```
public class PriorityQueue<E> implements Queue<E>
```

`PriorityQueue<E>()`	constructs an empty queue
`add(E value)`	adds **value** in sorted order to the queue
`peek()`	returns minimum element in queue
`remove()`	removes/returns minimum element in queue
`size()`	returns the number of elements in queue

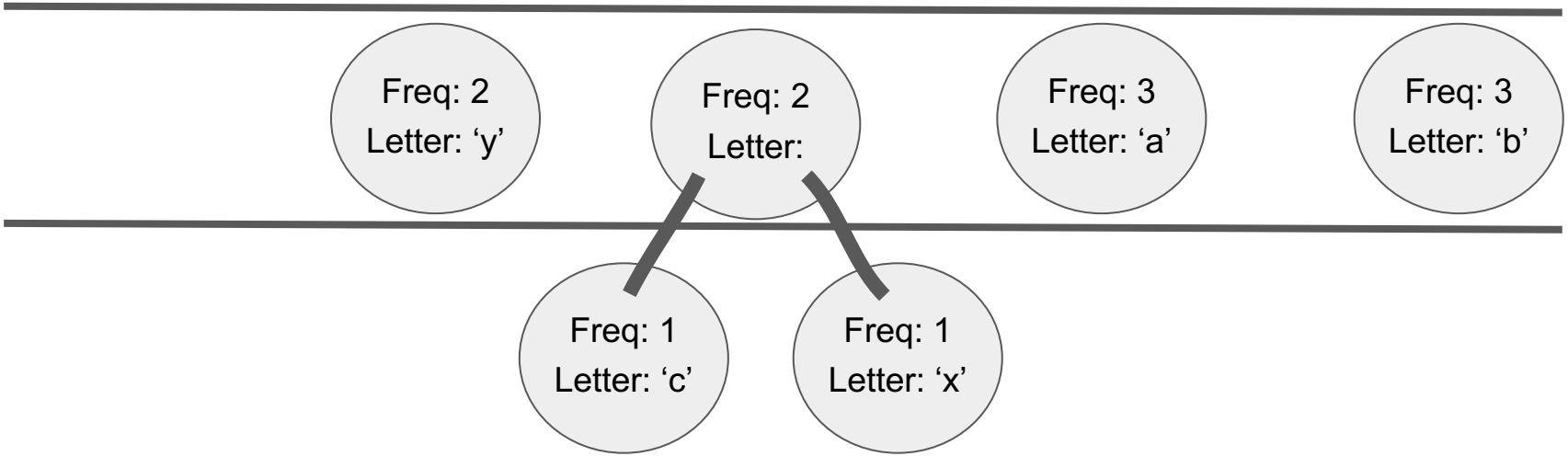
```
Queue<String> tas = new PriorityQueue<String>();
tas.add("Watson");
tas.add("Sherlock");
tas.remove(); // "Sherlock"
```

comPable

Making a HuffmanCode

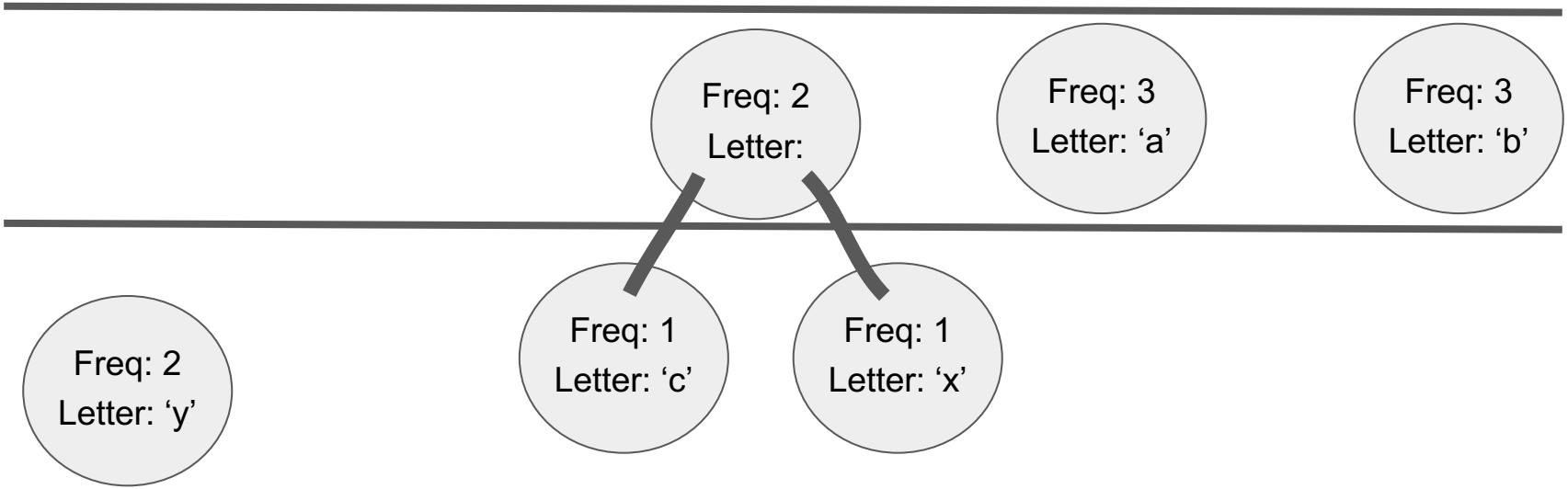
Extracted Letters:

c x yy aaa bbb



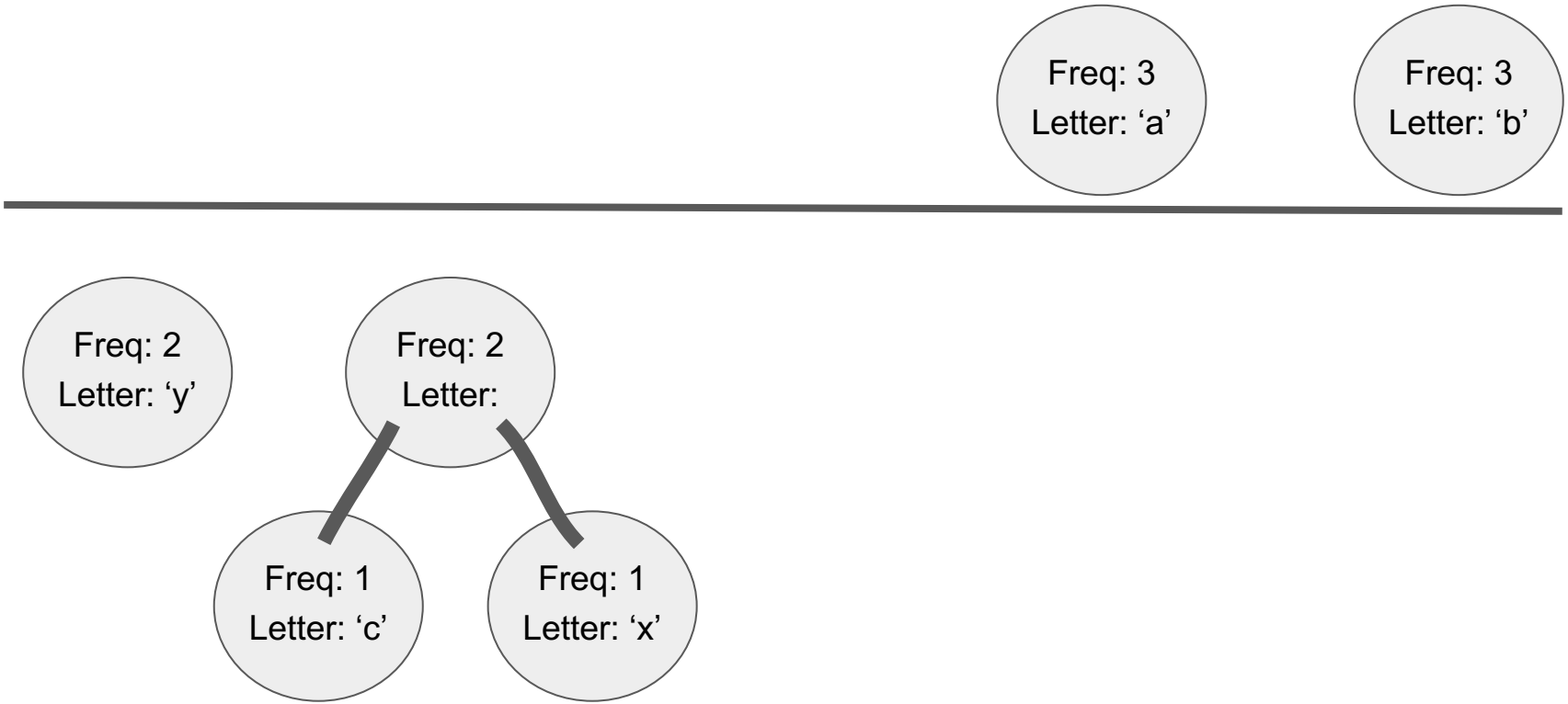
Making a HuffmanCode

Extracted Letters:
c x yy aaa bbb



Making a HuffmanCode

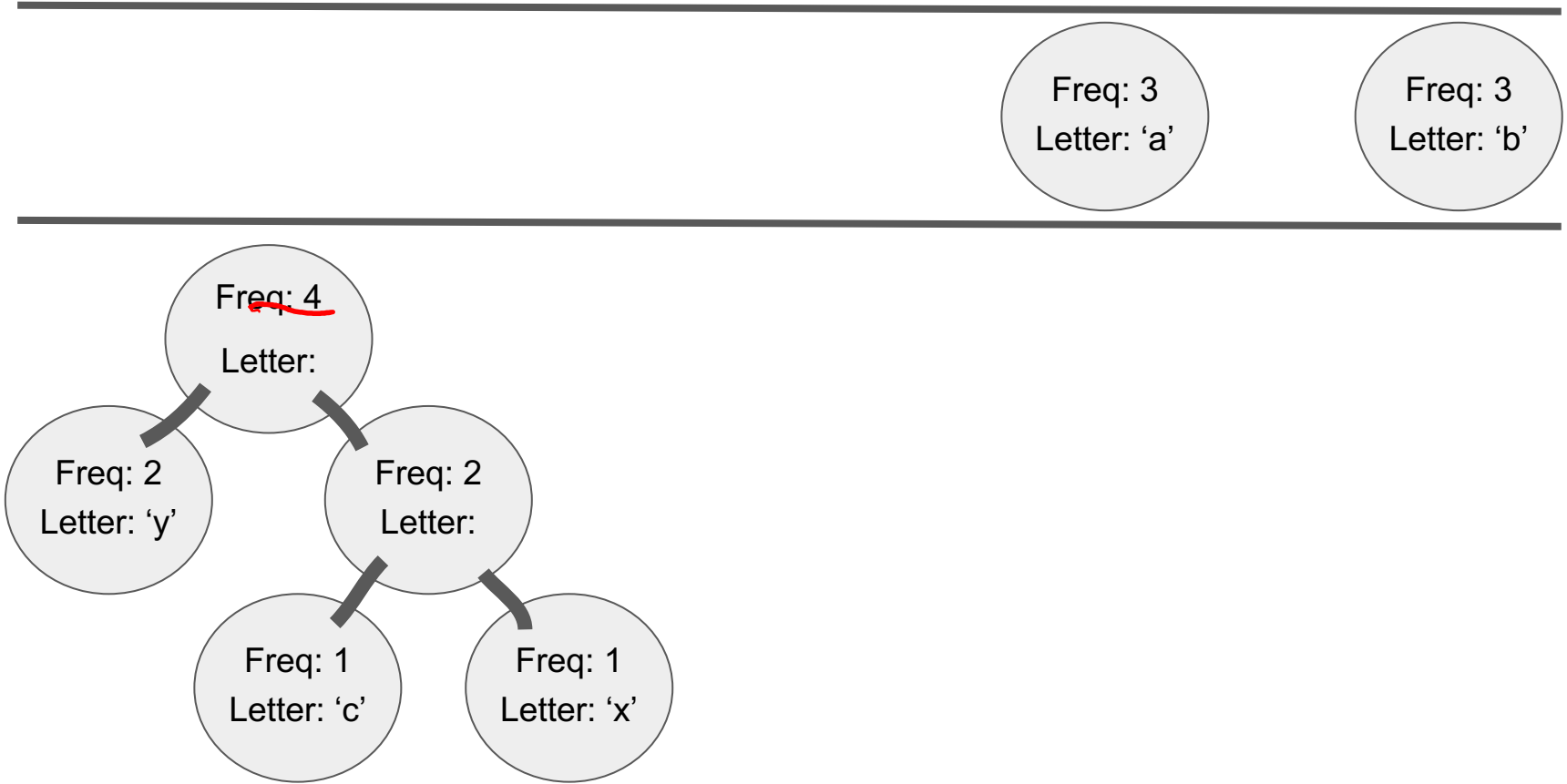
Extracted Letters:
c x yy aaa bbb



Making a HuffmanCode

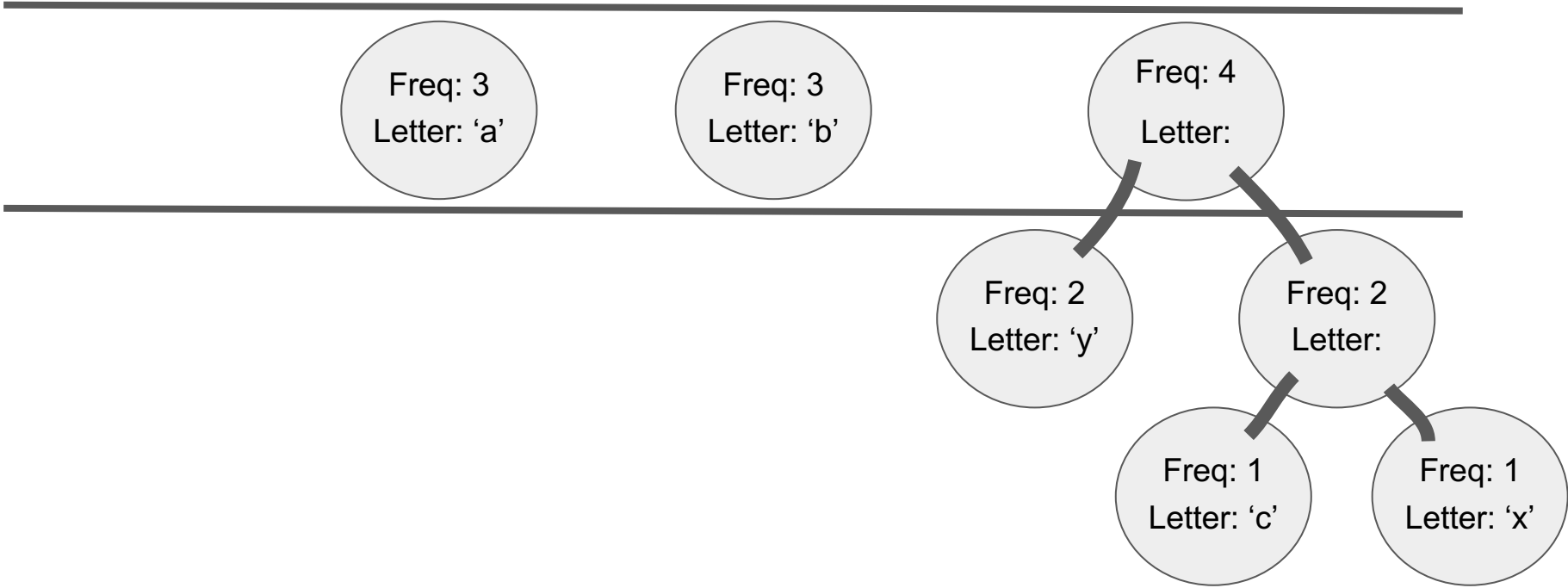
Extracted Letters:

c x yy aaa bbb



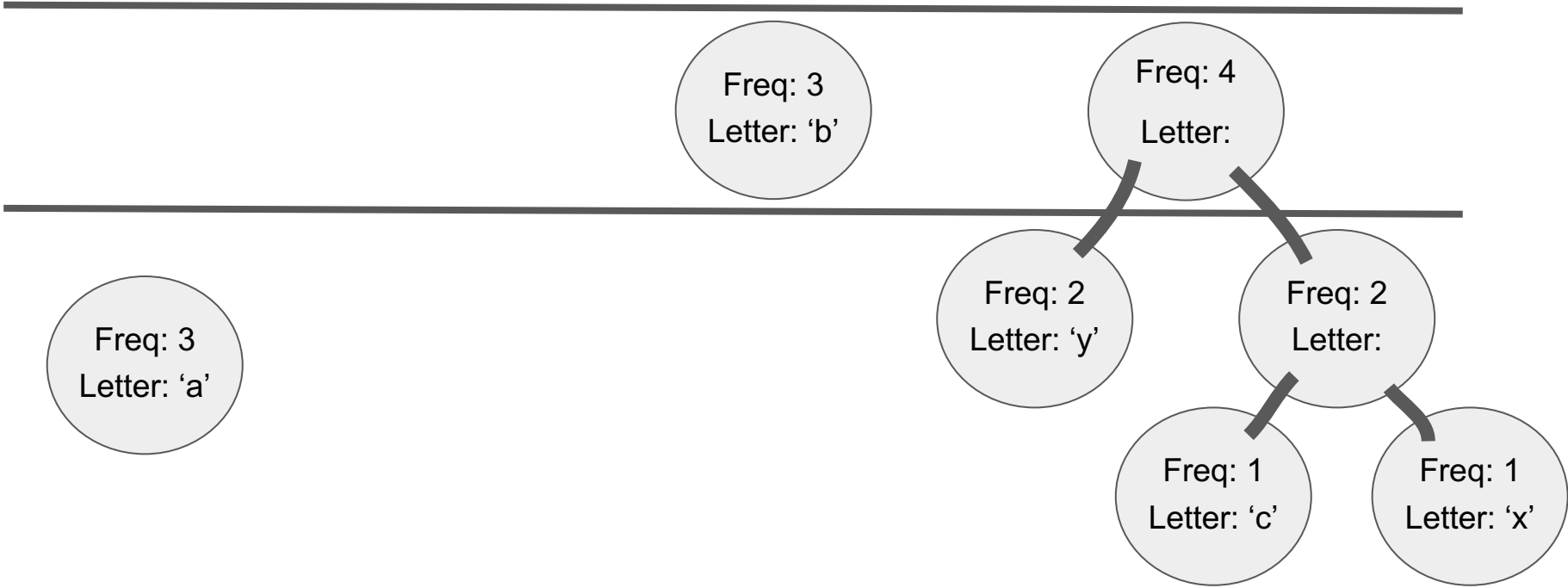
Making a HuffmanCode

Extracted Letters:
c x yy aaa bbb



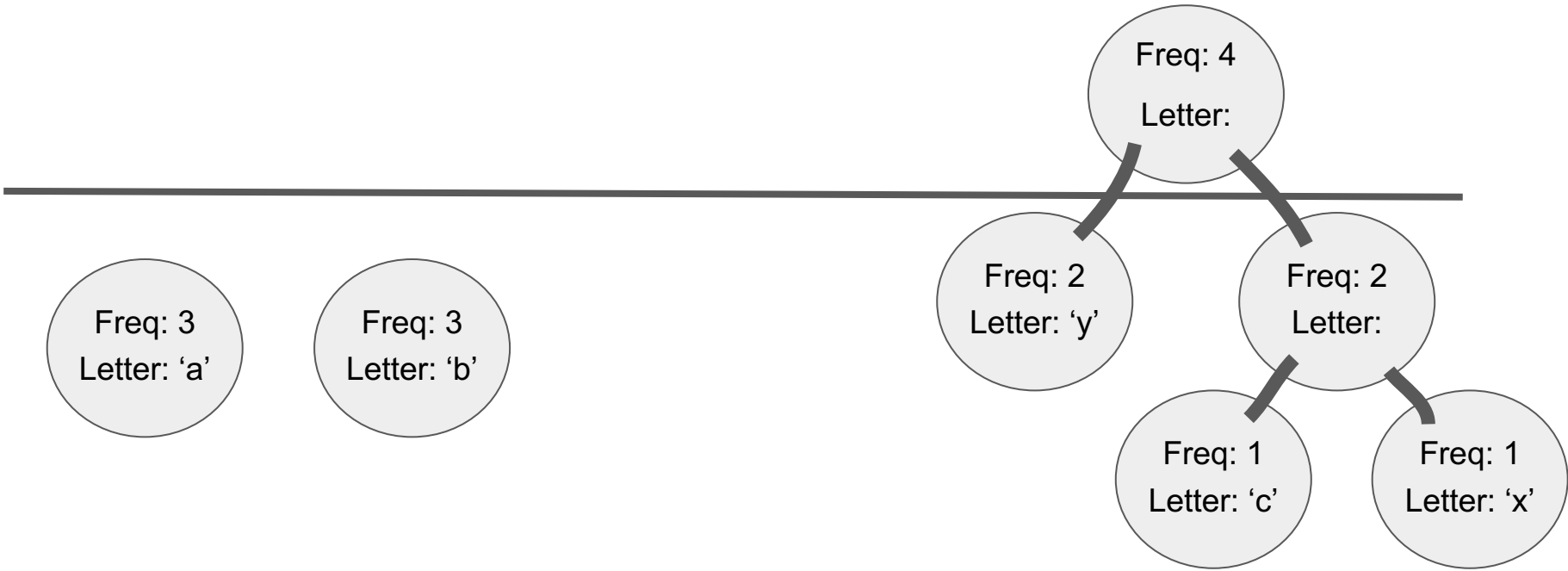
Making a HuffmanCode

Extracted Letters:
c x yy aaa bbb



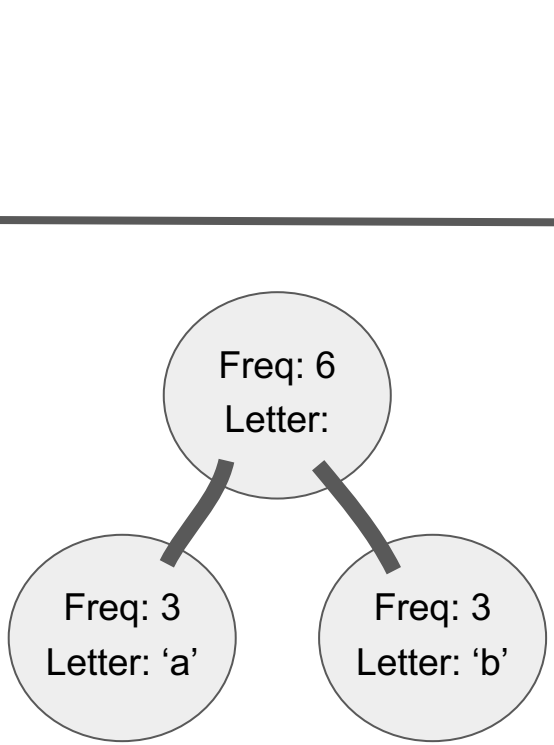
Making a HuffmanCode

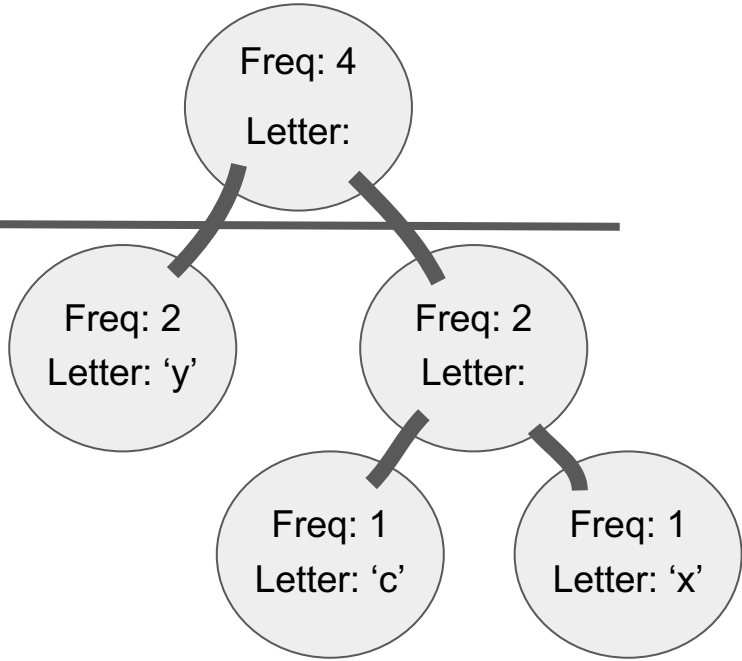
Extracted Letters:
c x yy aaa bbb



Making a HuffmanCode

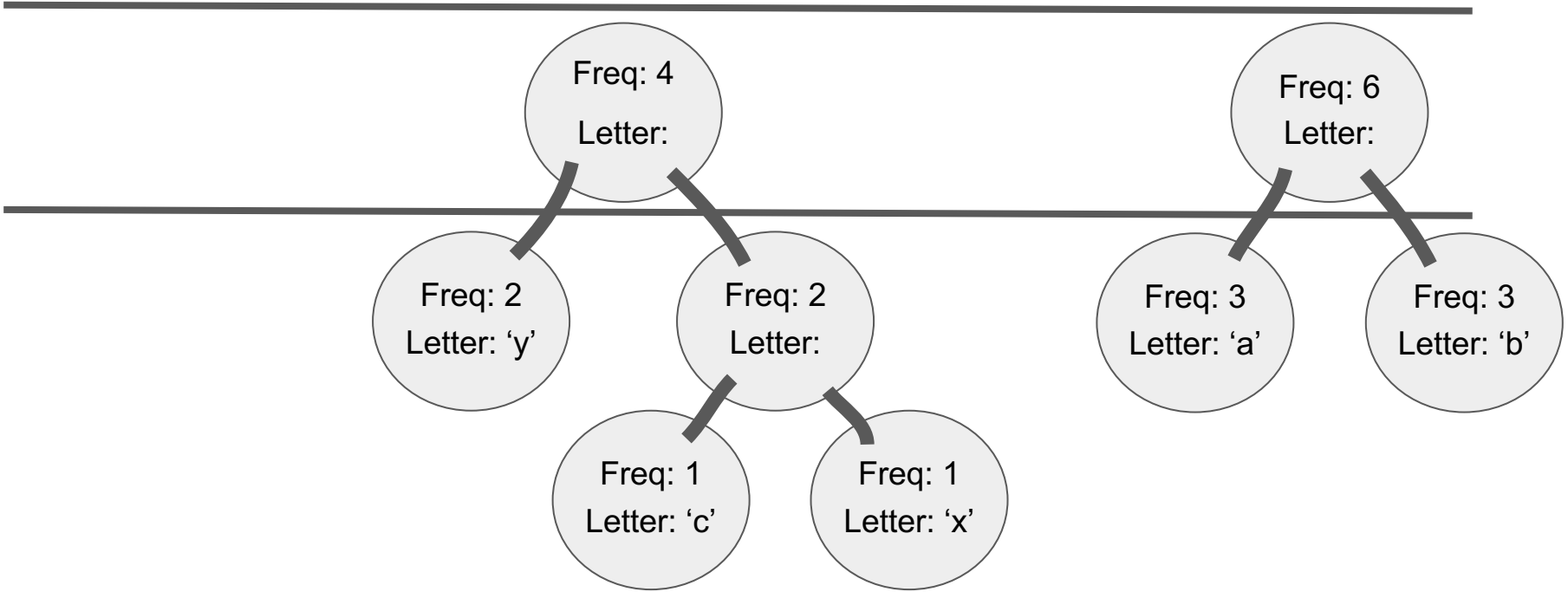
Extracted Letters:
c x yy aaa bbb

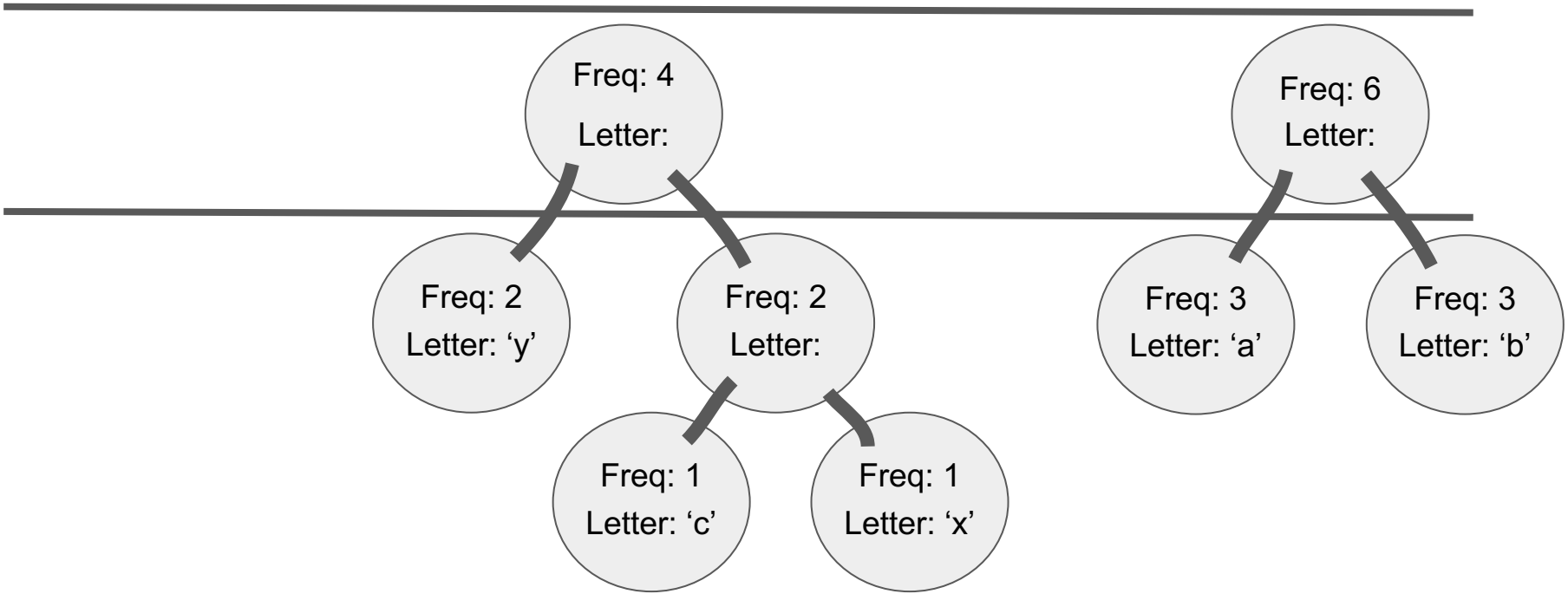


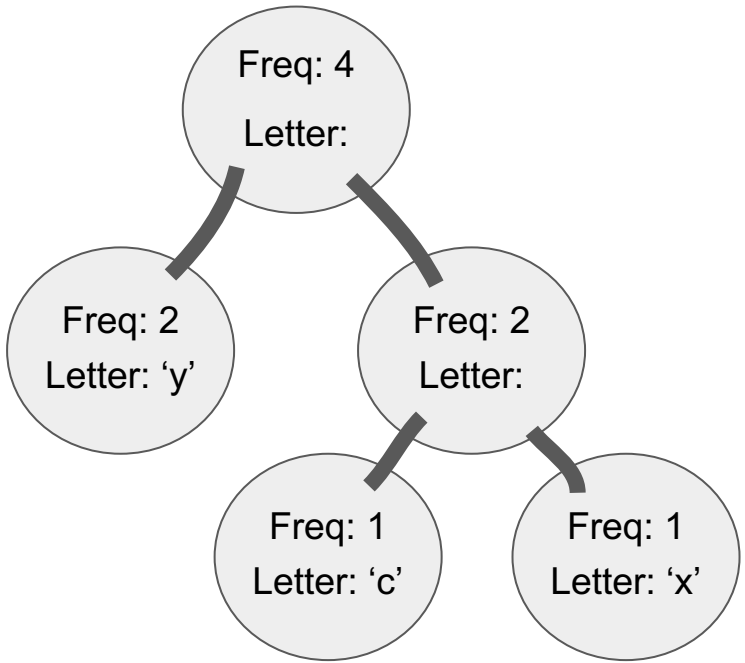


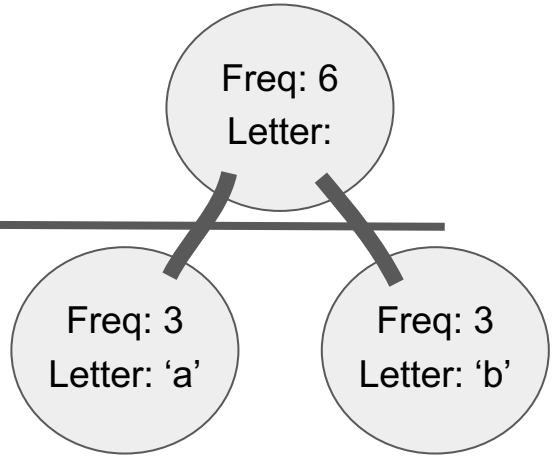
Making a HuffmanCode

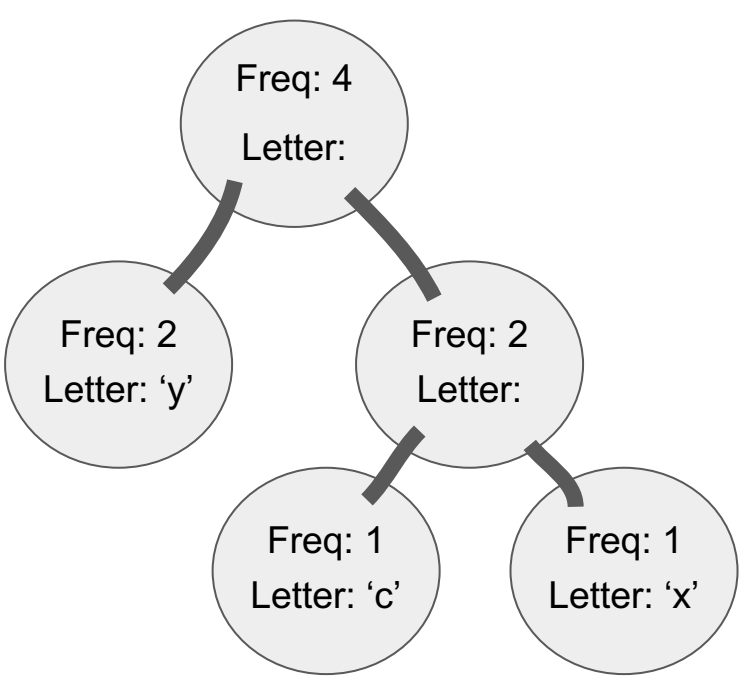
Extracted Letters:
c x yy aaa bbb

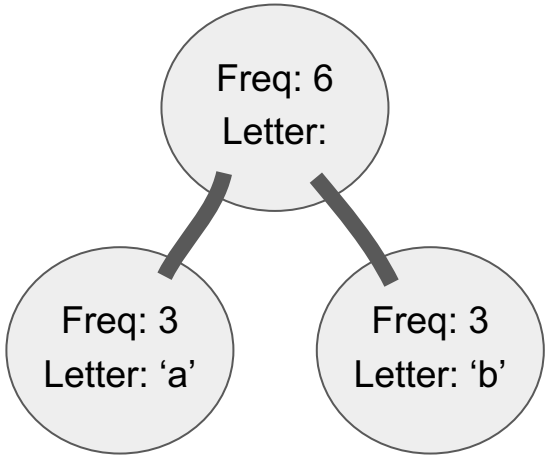


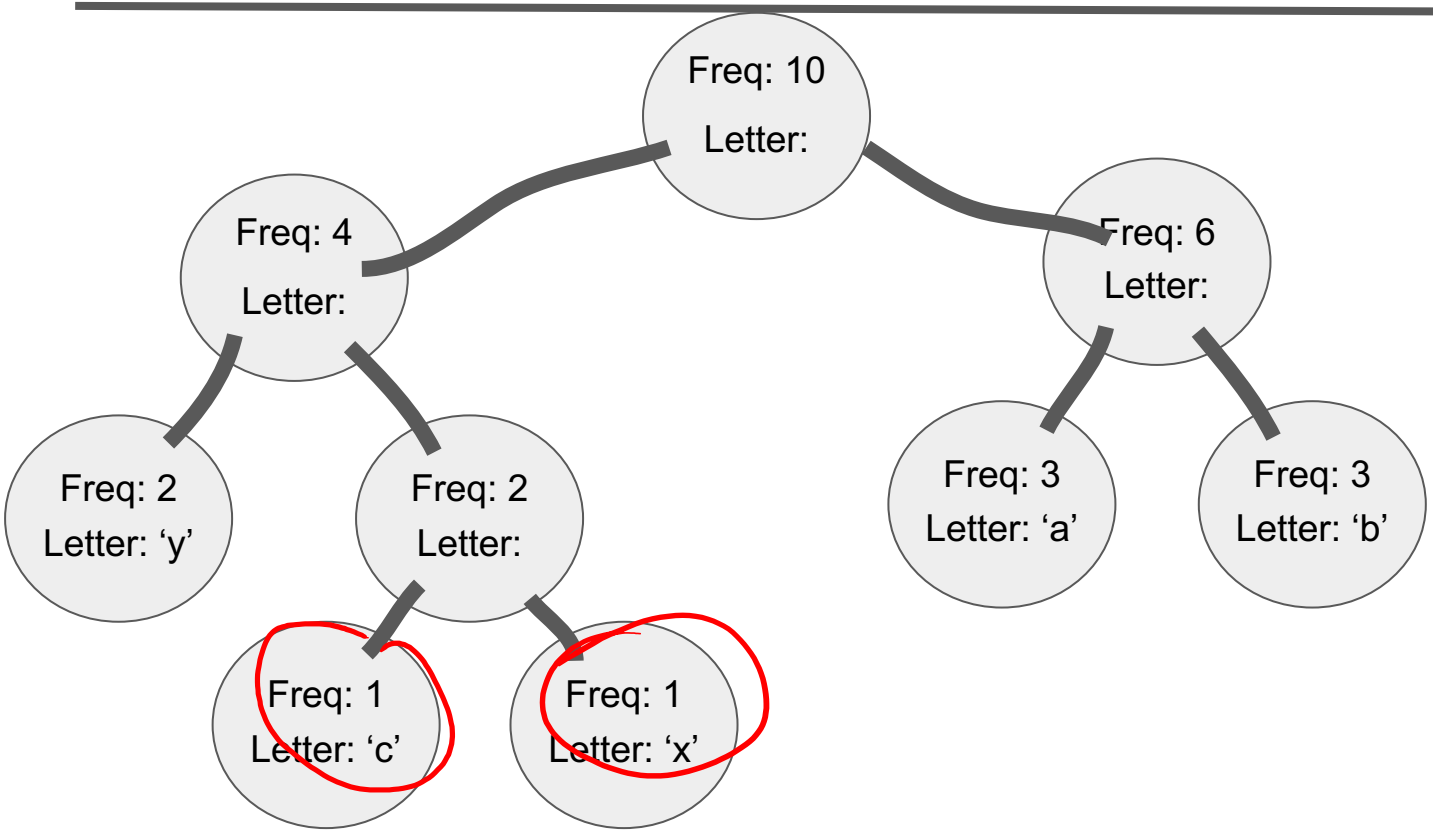

Freq: 4
Letter:
Freq: 6
Letter:
Freq: 2
Letter: ‘y’
Freq: 2
Letter:
Freq: 3
Letter: ‘a’
Freq: 3
Letter: ‘b’
Freq: 1
Letter: ‘c’
Freq: 1
Letter: ‘x’

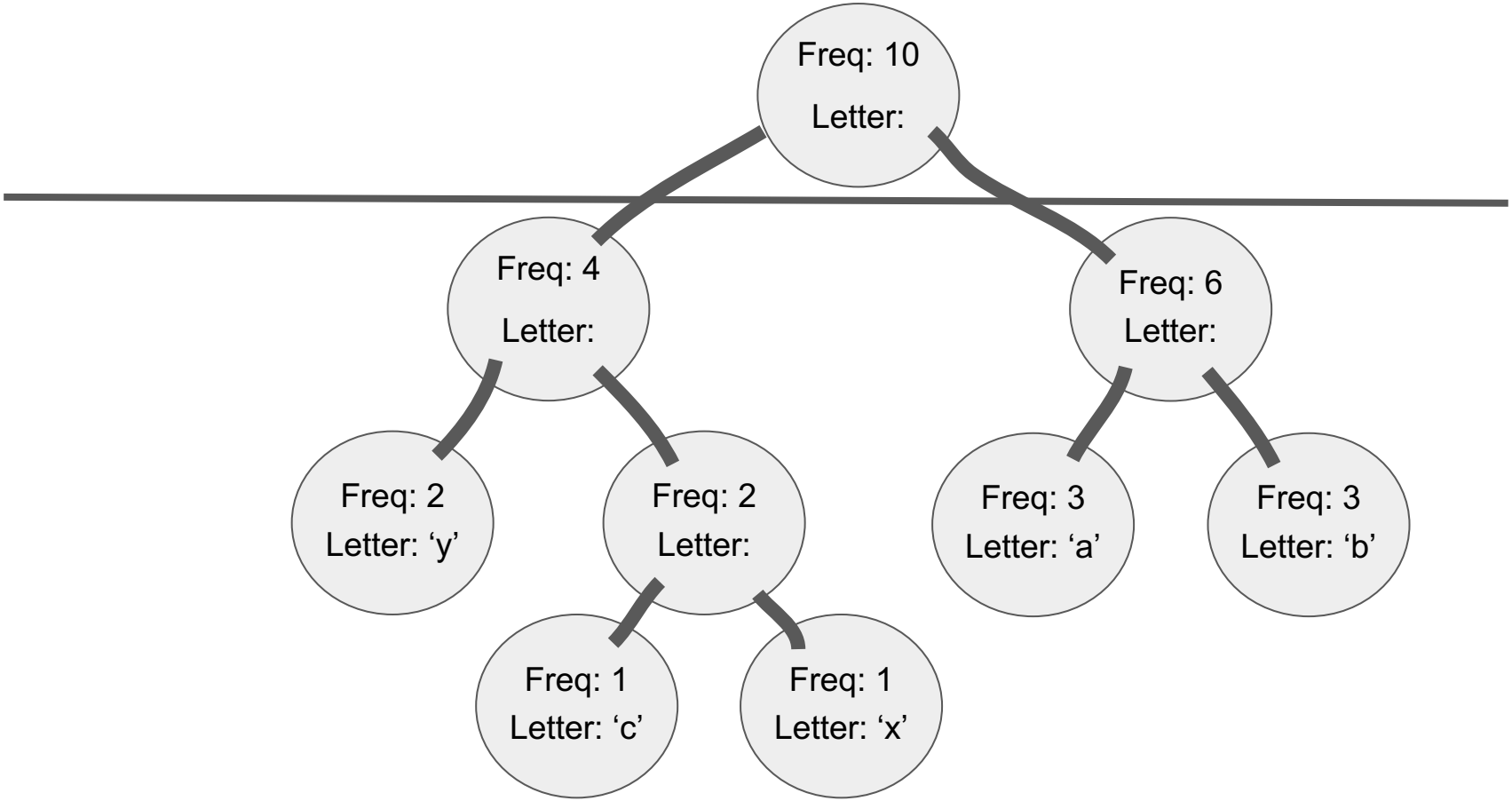

Freq: 4
Letter:
Freq: 2
Letter: ‘y’
Freq: 2
Letter:
Freq: 1
Letter: ‘c’
Freq: 1
Letter: ‘x’

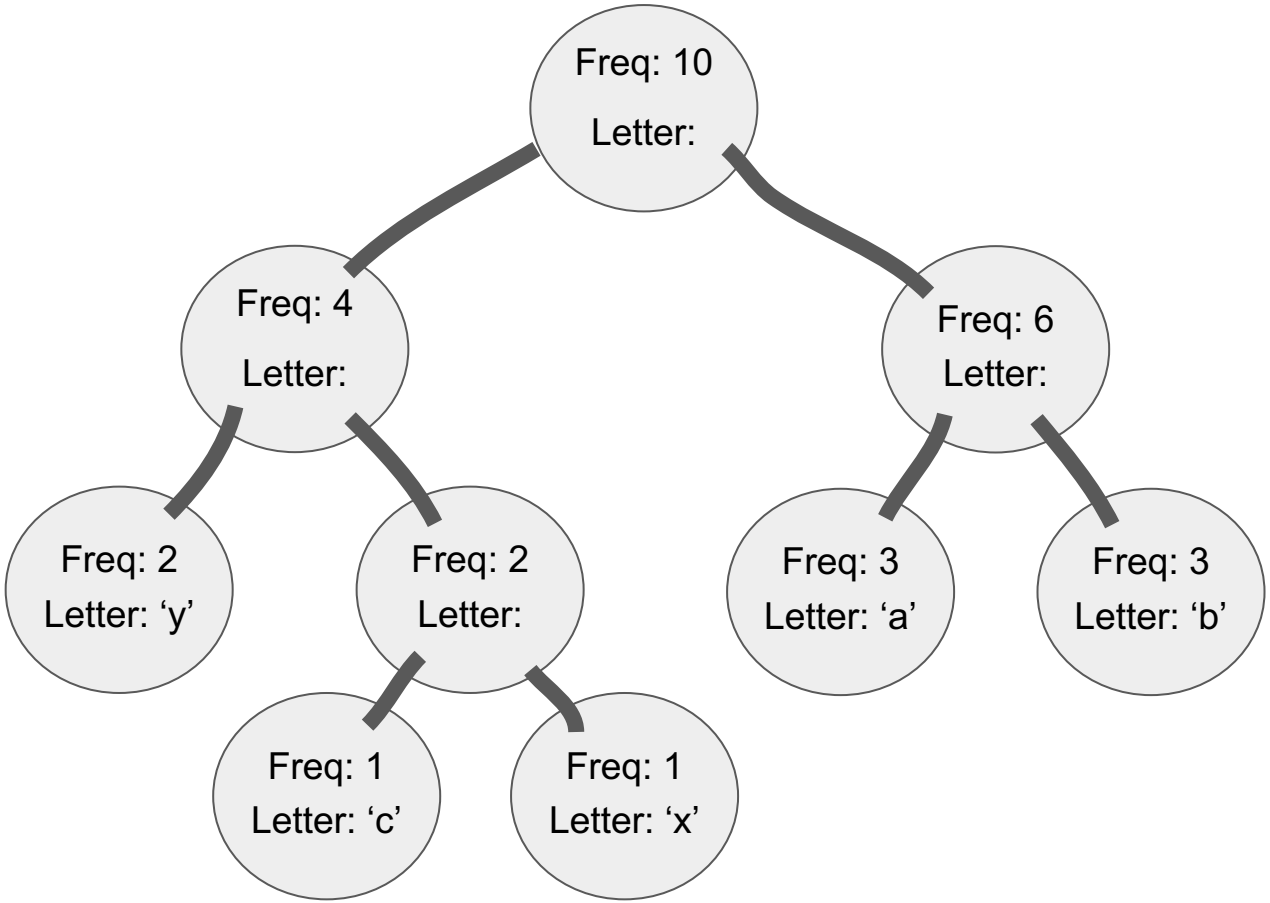

Freq: 6
Letter:
Freq: 3
Letter: ‘a’
Freq: 3
Letter: ‘b’


Freq: 4
Letter:
Freq: 2
Letter: ‘y’
Freq: 2
Letter:
Freq: 1
Letter: ‘c’
Freq: 1
Letter: ‘x’


Freq: 6
Letter:
Freq: 3
Letter: ‘a’
Freq: 3
Letter: ‘b’

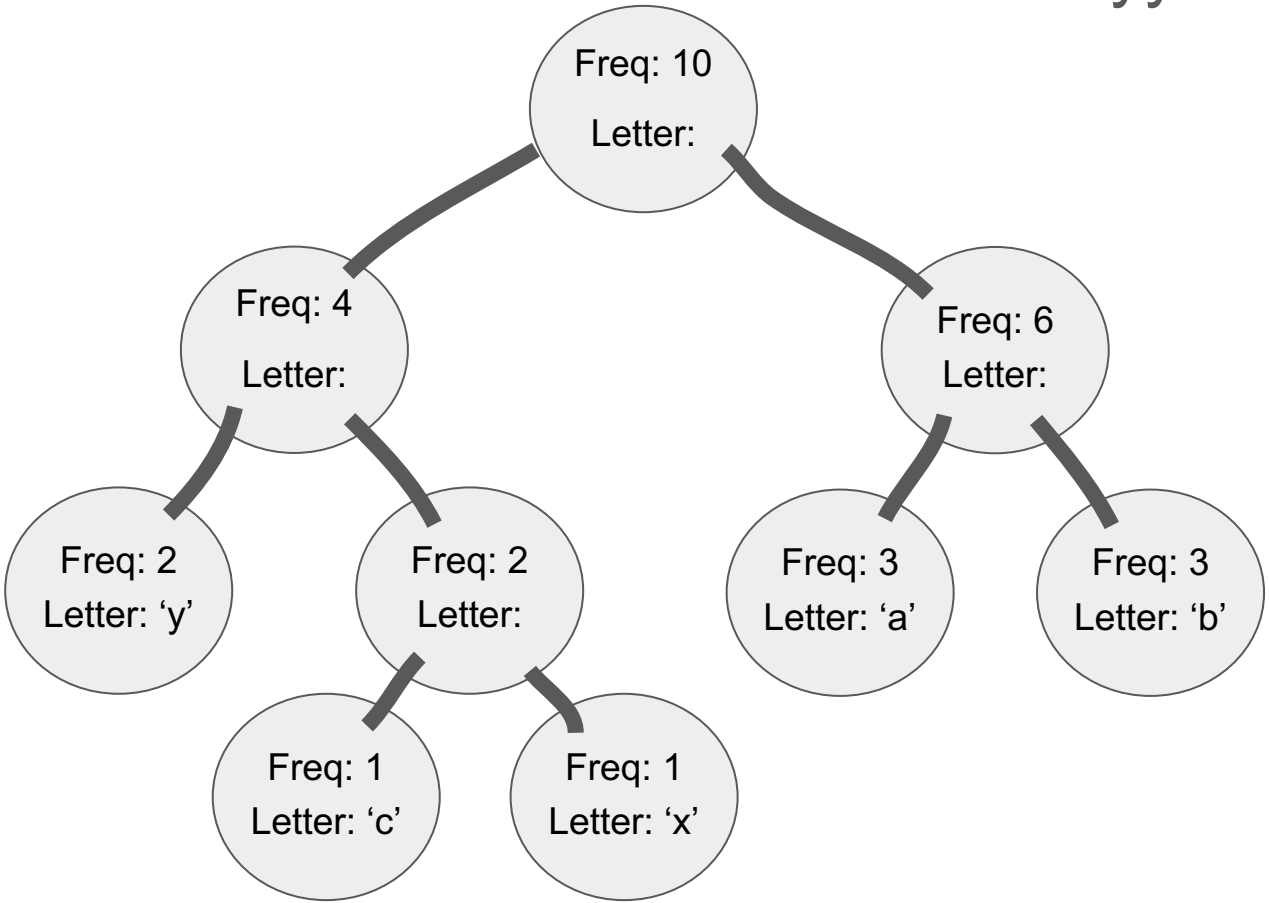

Freq: 10
Letter:
Freq: 4
Letter:
Freq: 6
Letter:
Freq: 2
Letter: ‘y’
Freq: 2
Letter:
Freq: 3
Letter: ‘a’
Freq: 3
Letter: ‘b’
Freq: 1
Letter: ‘c’
Freq: 1
Letter: ‘x’


Freq: 10
Letter:
Freq: 4
Letter:
Freq: 6
Letter:
Freq: 2
Letter: ‘y’
Freq: 2
Letter:
Freq: 3
Letter: ‘a’
Freq: 3
Letter: ‘b’
Freq: 1
Letter: ‘c’
Freq: 1
Letter: ‘x’


Freq: 10
Letter:
Freq: 4
Letter:
Freq: 6
Letter:
Freq: 2
Letter: ‘y’
Freq: 2
Letter:
Freq: 3
Letter: ‘a’
Freq: 3
Letter: ‘b’
Freq: 1
Letter: ‘c’
Freq: 1
Letter: ‘x’

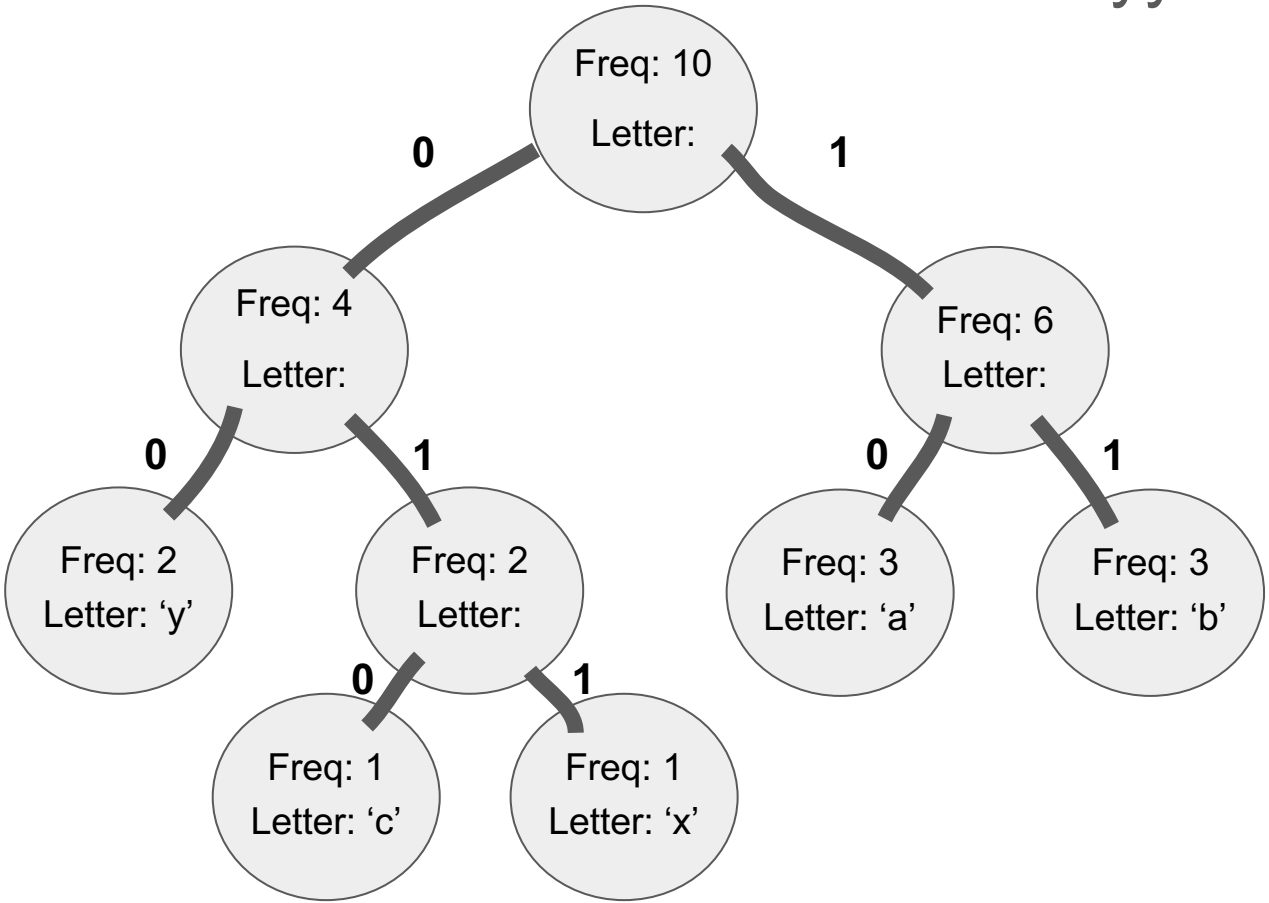
Making a HuffmanCode

Extracted Letters:
c x yy aaa bbb



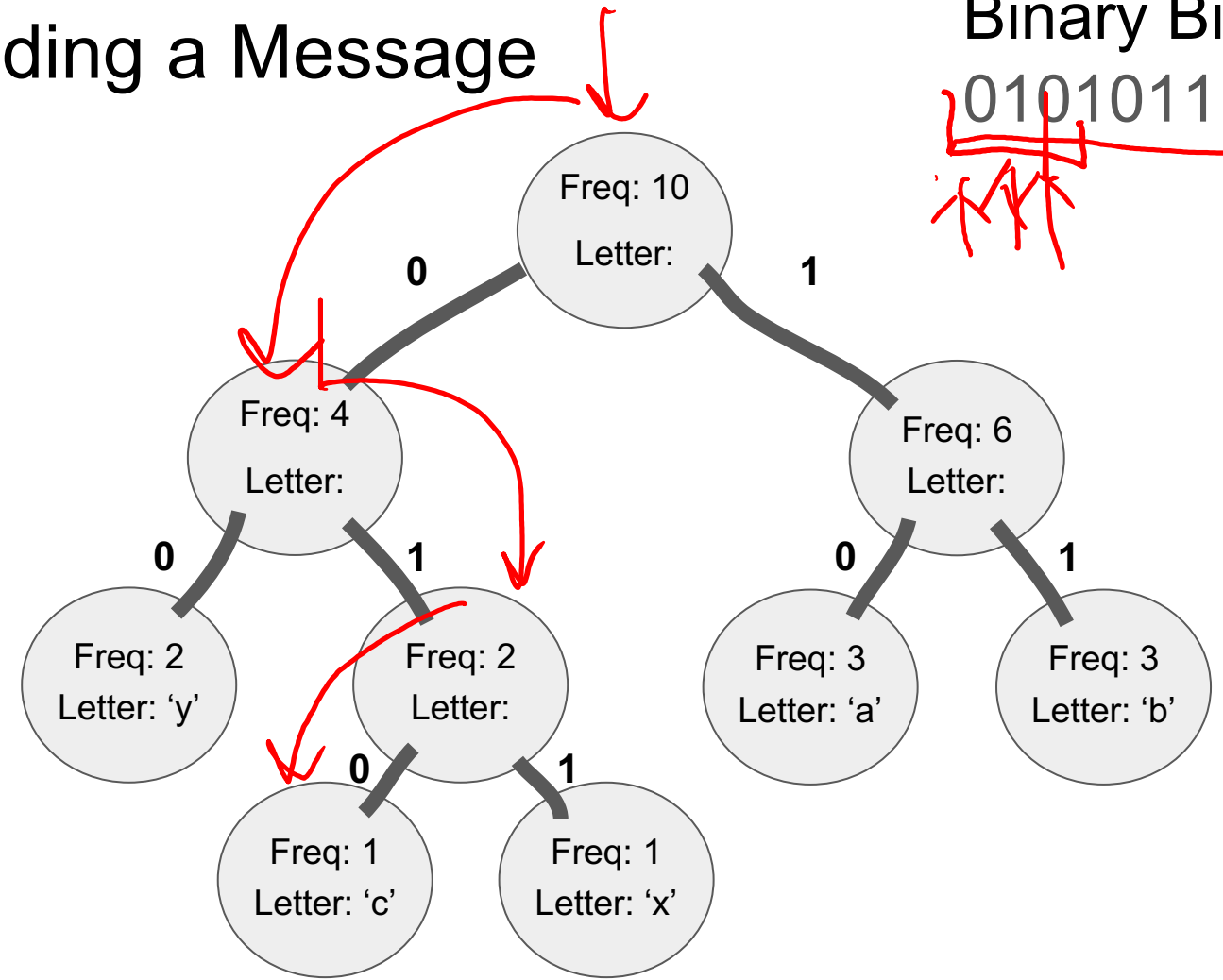
Making a HuffmanCode

Extracted Letters:
c x yy aaa bbb



Decoding a Message

Binary Bits:
0101101111

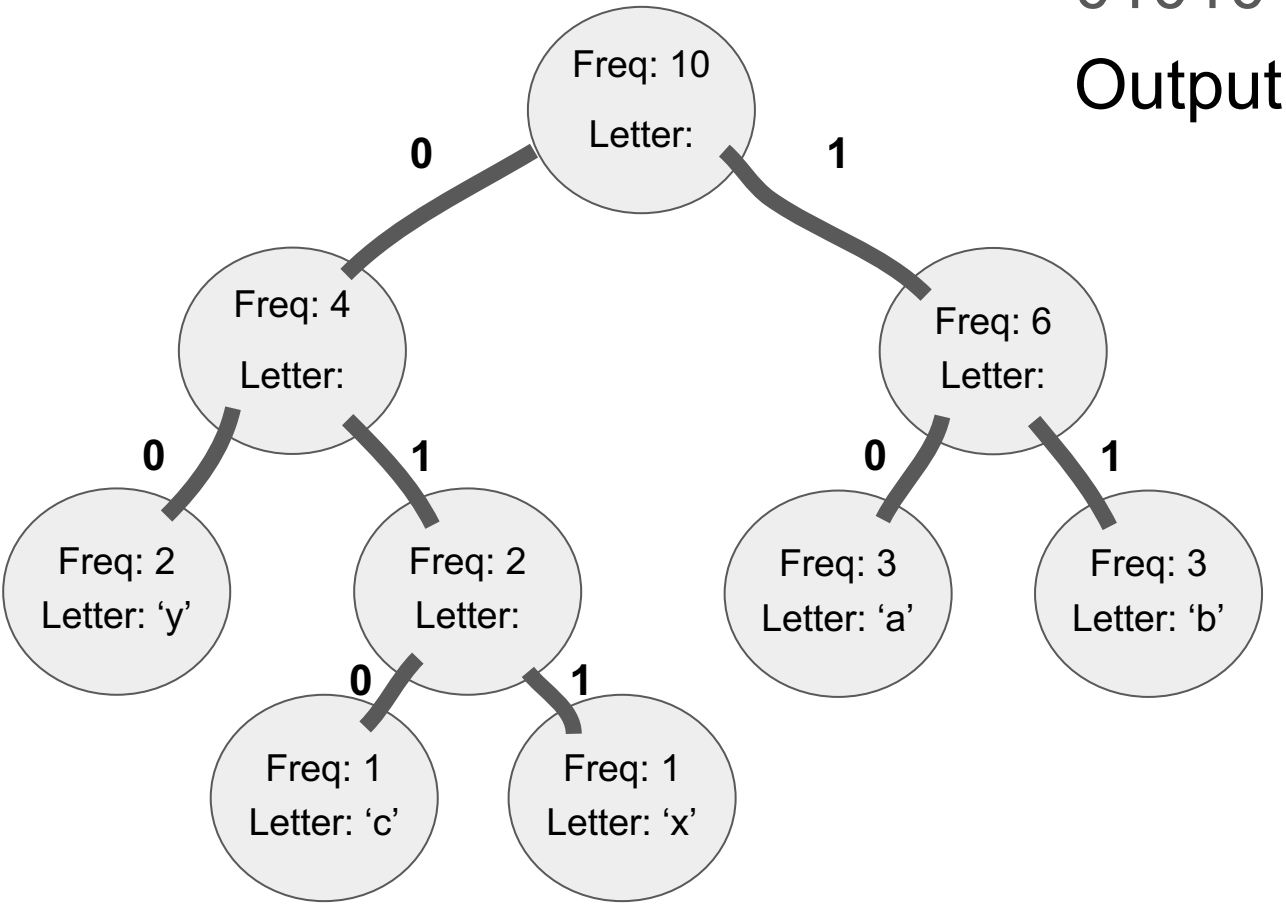


Decoding a Message

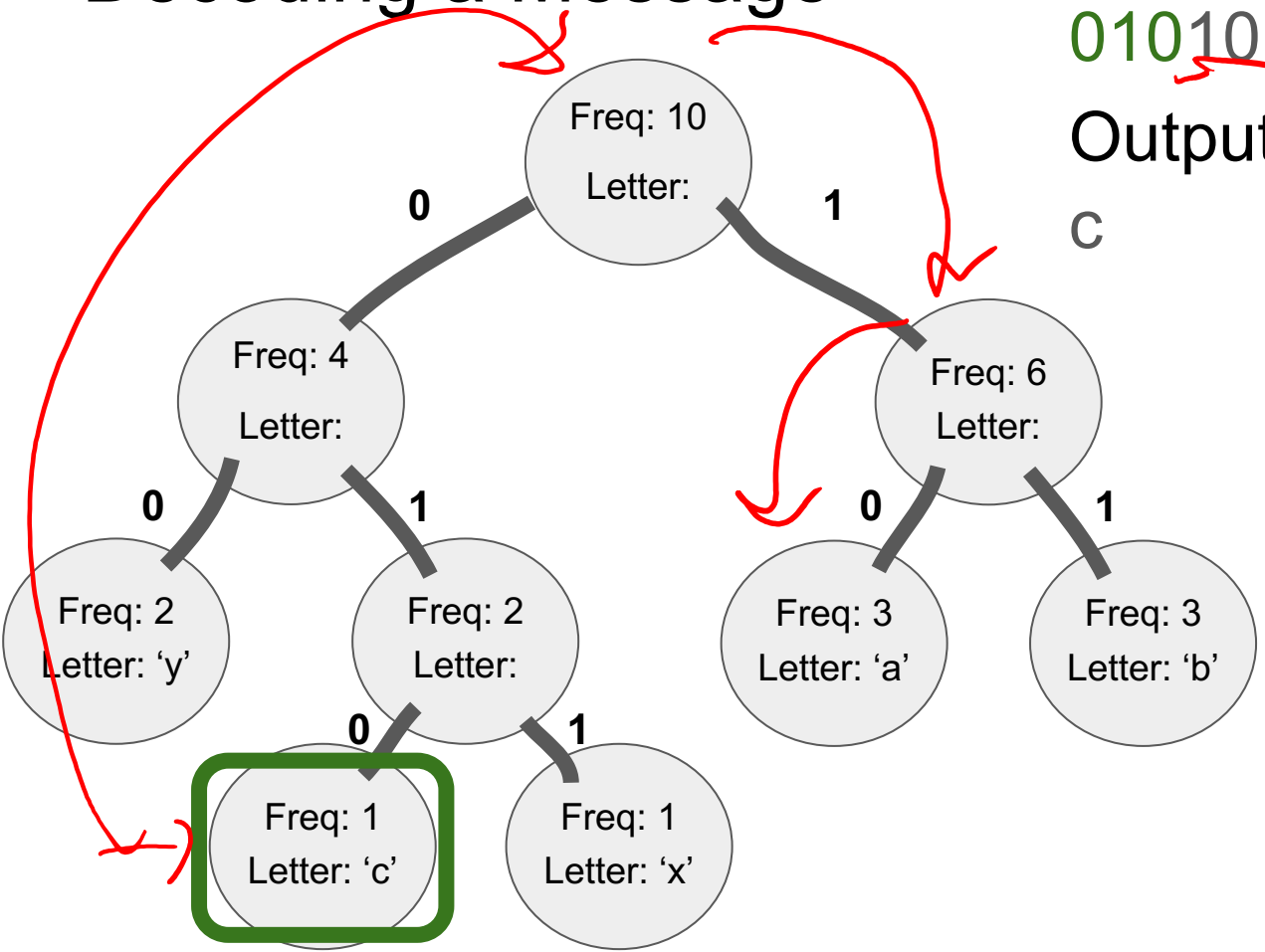
Binary Bits:

0101011111

Output:



Decoding a Message



Binary Bits:

0101101111

Output:

c

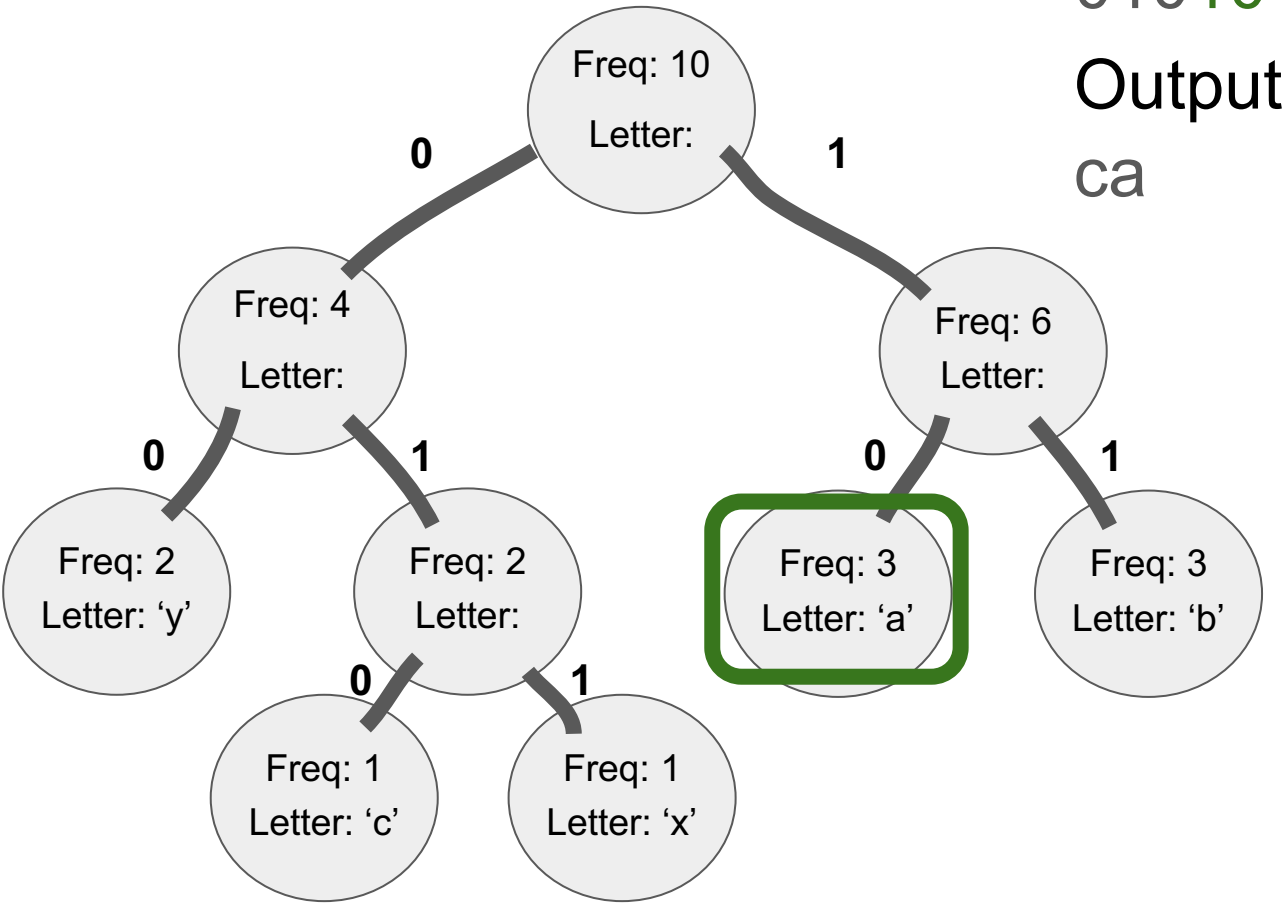
Decoding a Message

Binary Bits:

0101011111

Output:

ca



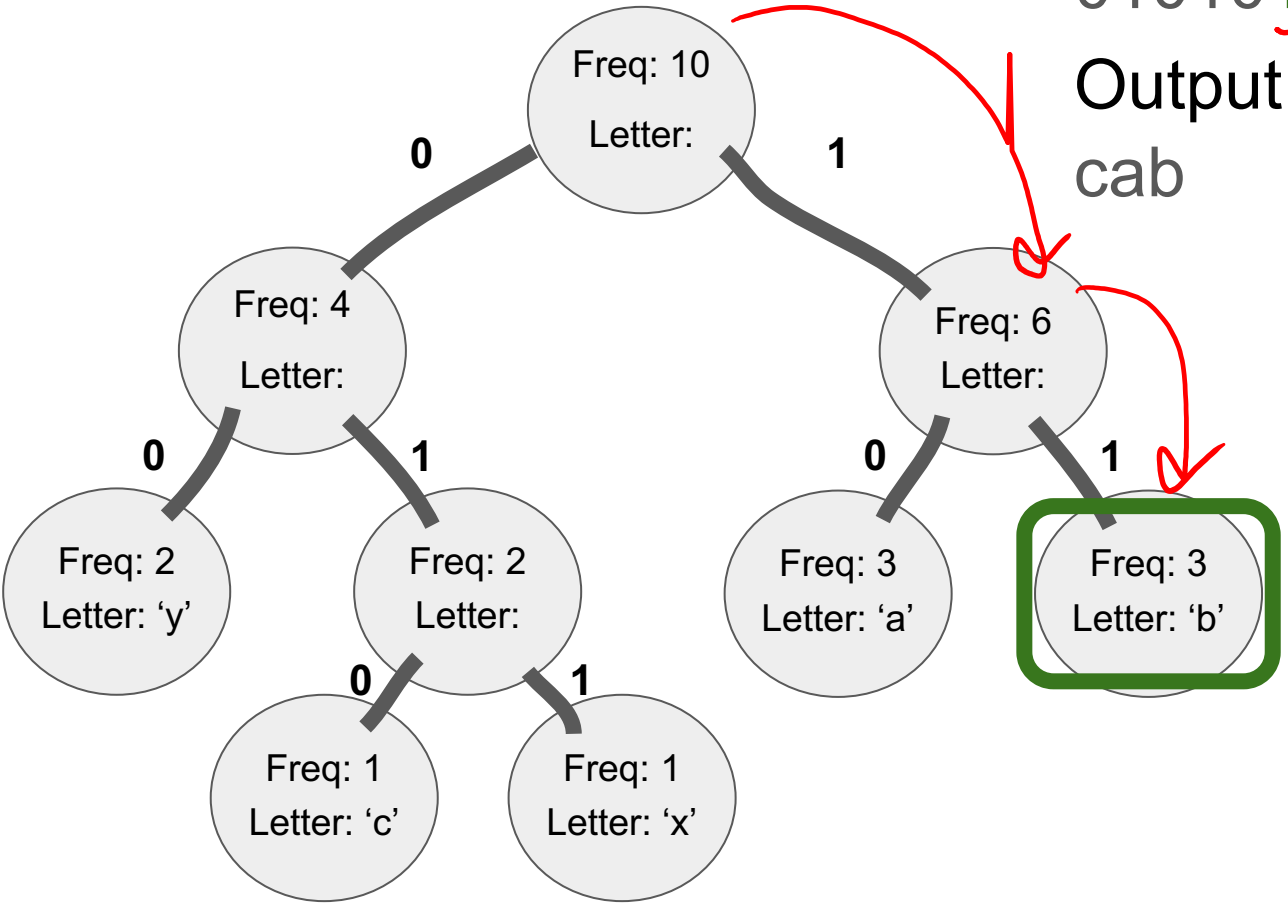
Decoding a Message

Binary Bits:

0101011111

Output:

cab



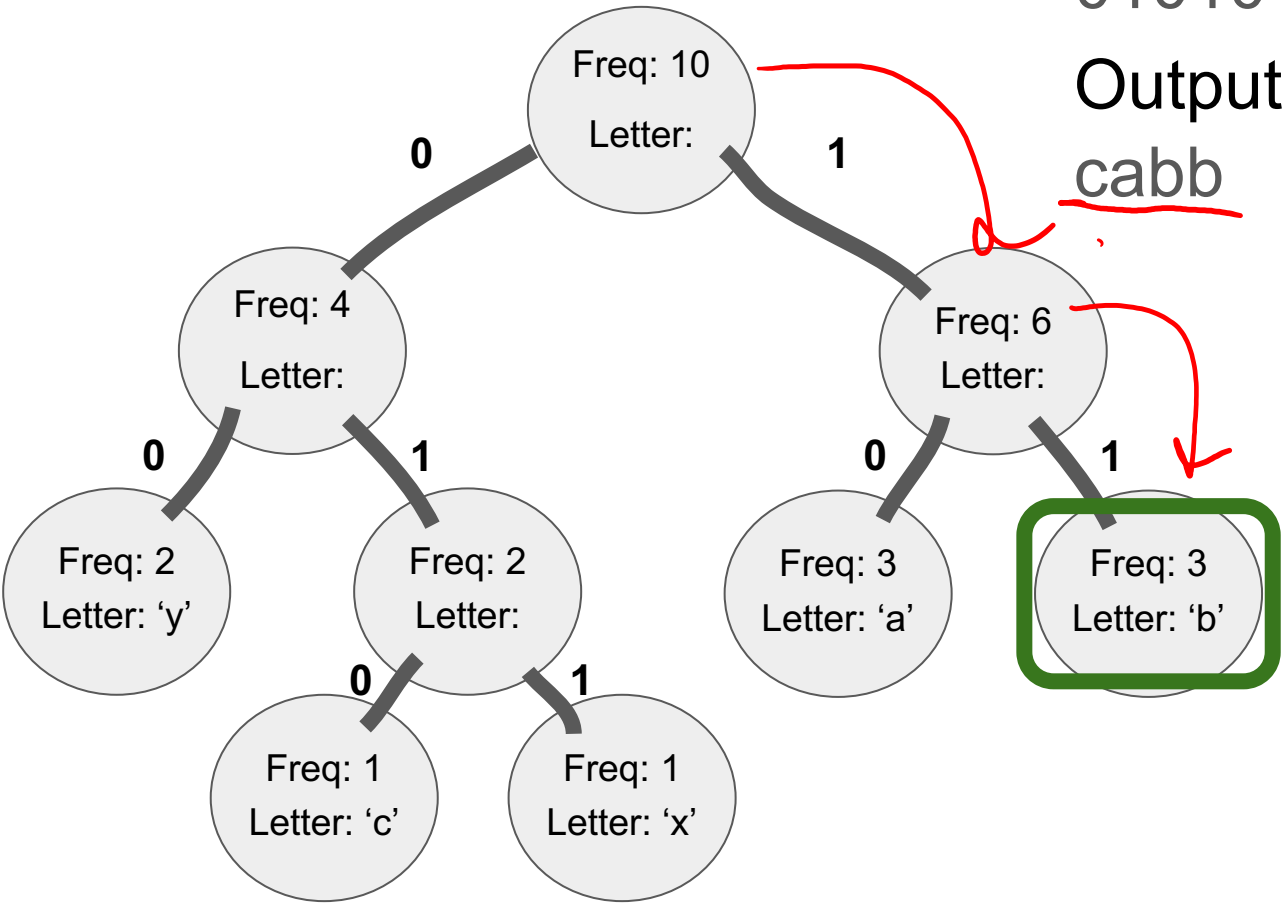
Decoding a Message

Binary Bits:

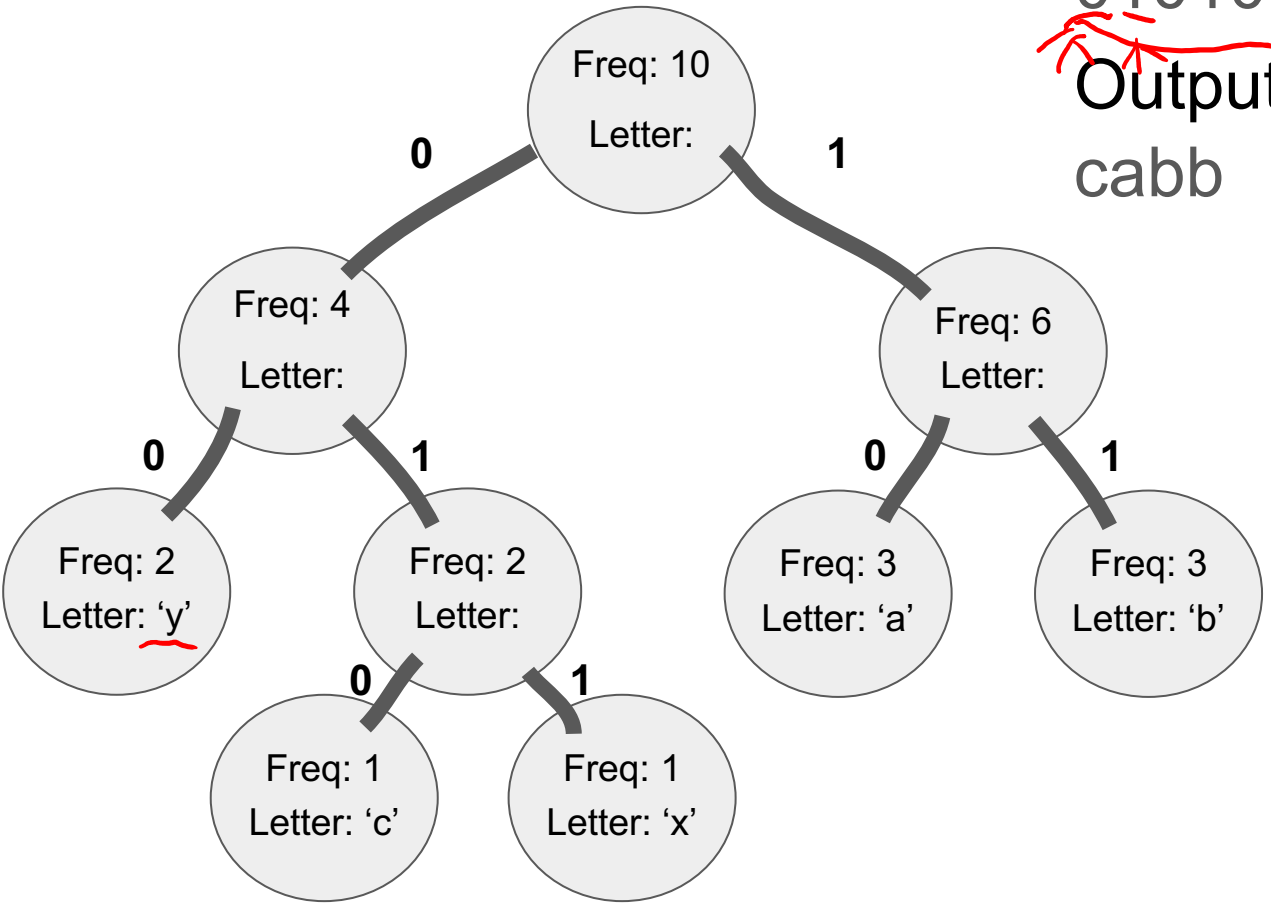
0101011 11

Output:

cabb



Decoding a Message

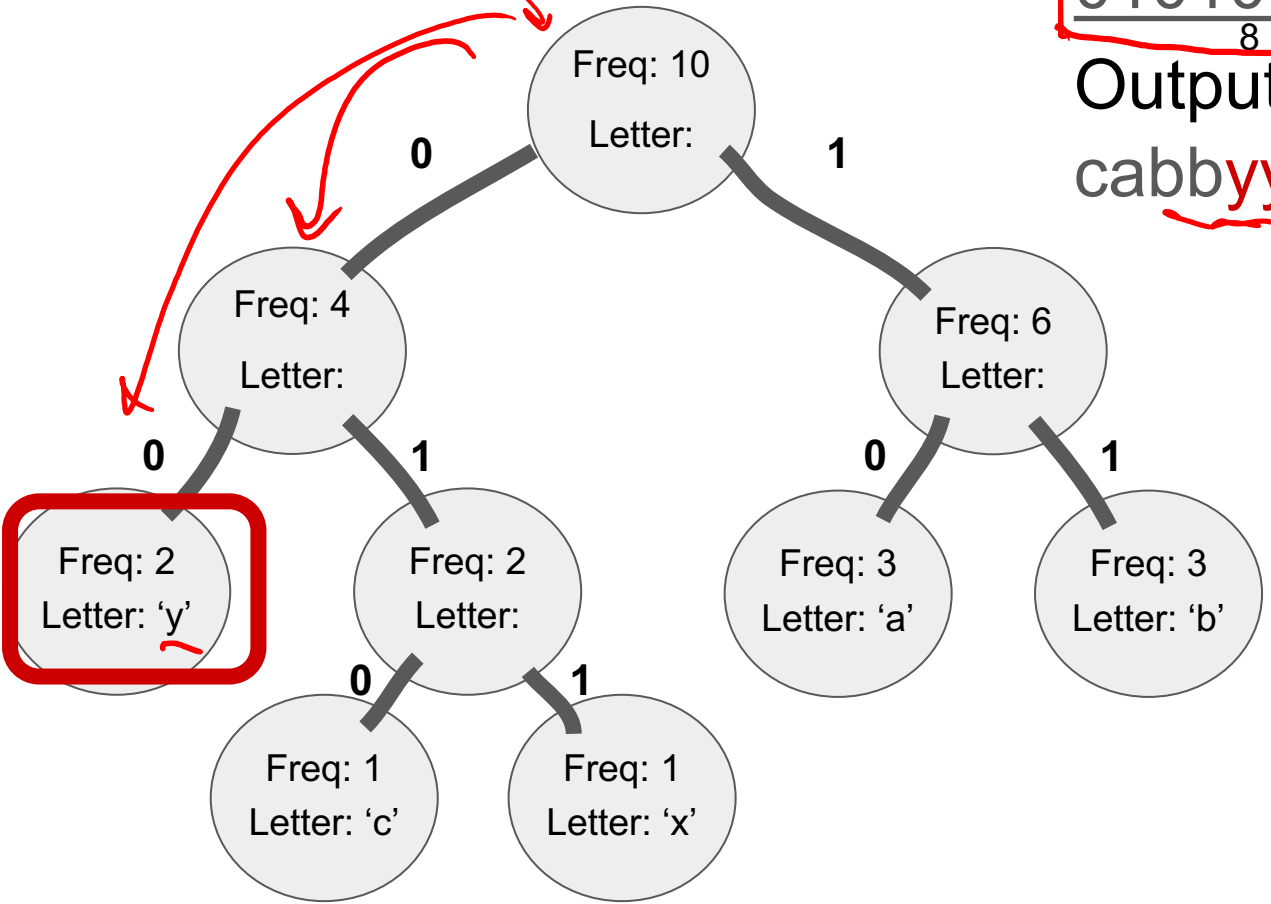


Binary Bits:
0101011111

Output:
cabb

BitInputStream:
Binary Bits, not Ascii
1’s and 0’s

Decoding a Message



Binary Bits:

01010111 10000000

8 8

Output:

cabbyyy?

Problem:

Can only write characters, which are 8 bits long

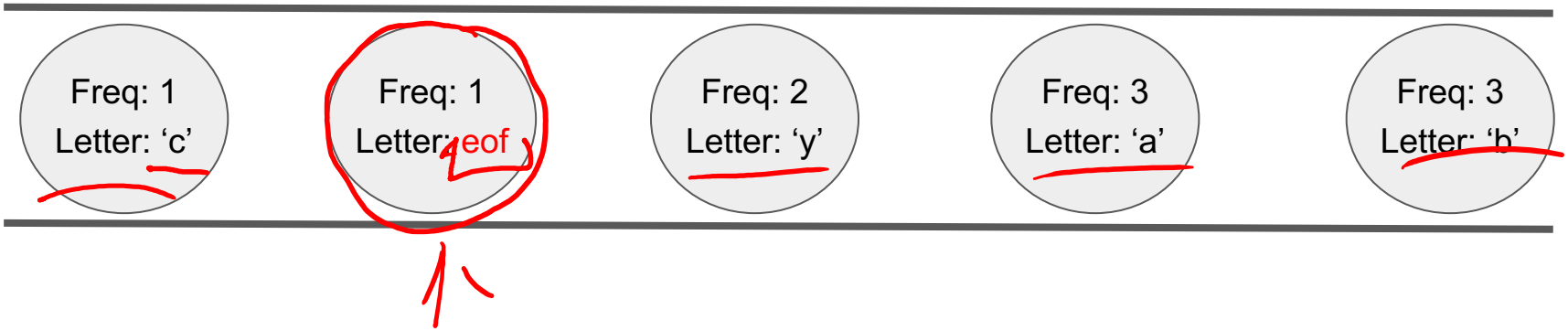
Solution:
Making a HuffmanCode - Example with Psudo EOF

Text:
babyyacab

Extracted Letters:
c yy aaa bbb

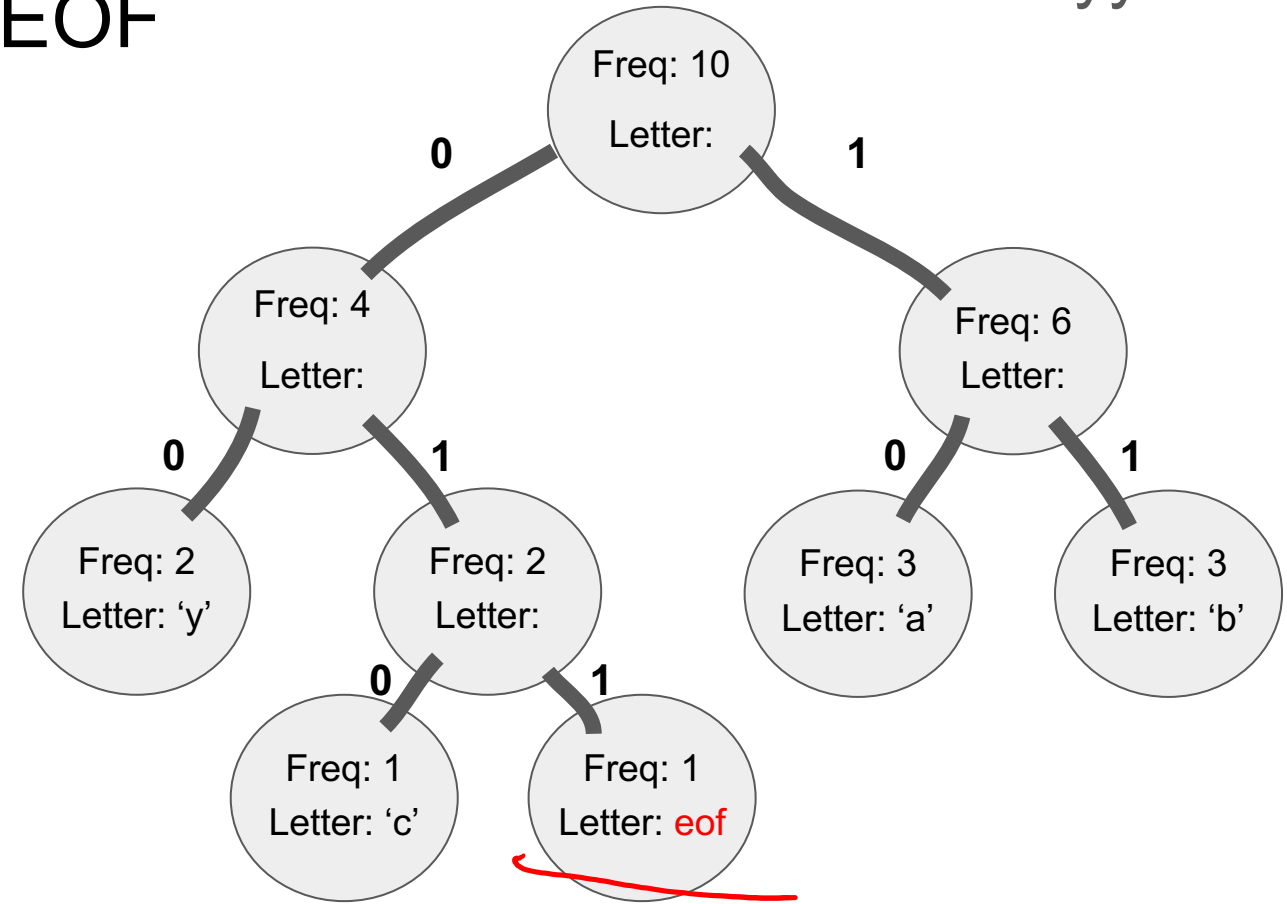
Making a HuffmanCode With EOF

Extracted Letters:
c yy aaa bbb



Making a HuffmanCode With EOF

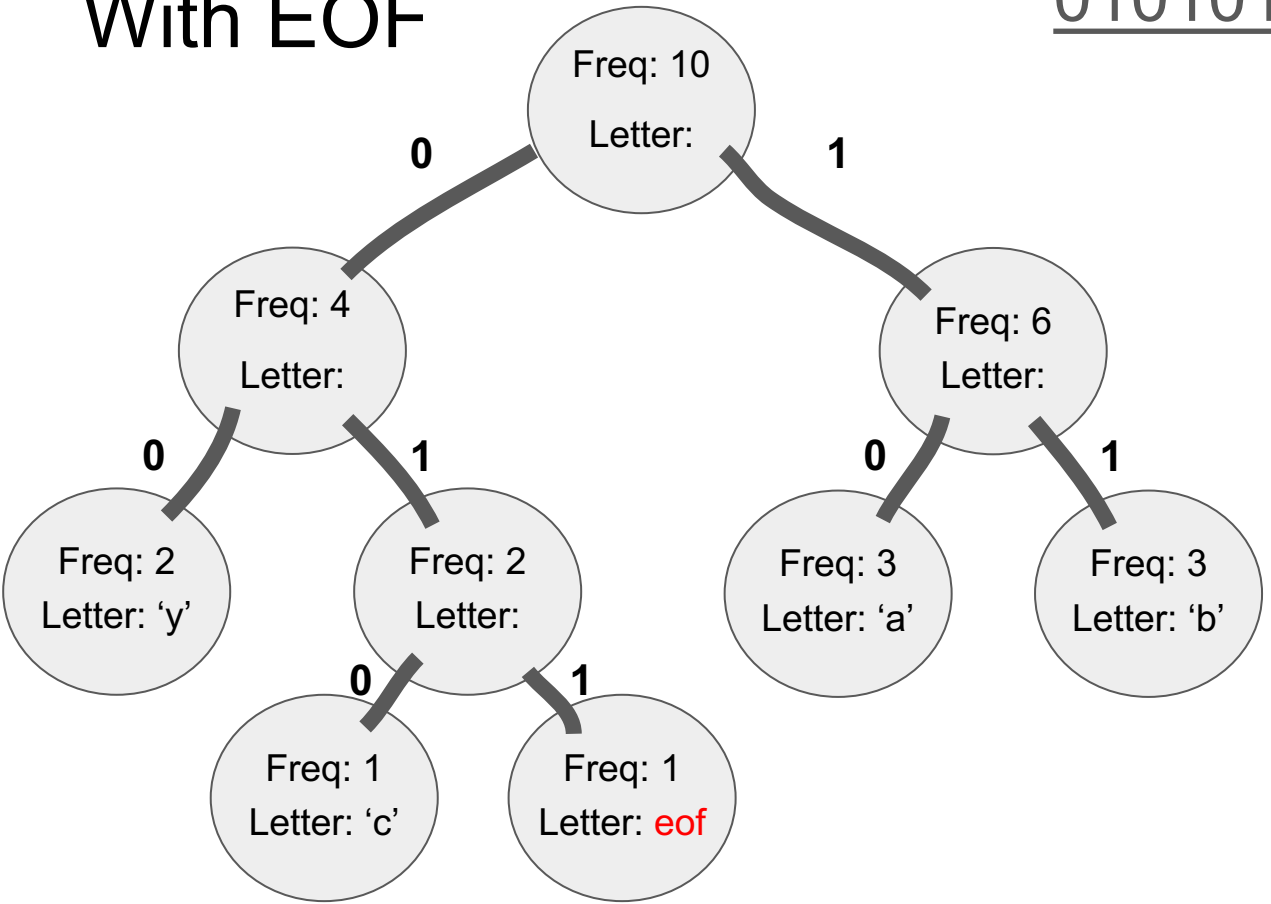
Extracted Letters:
c yy aaa bbb



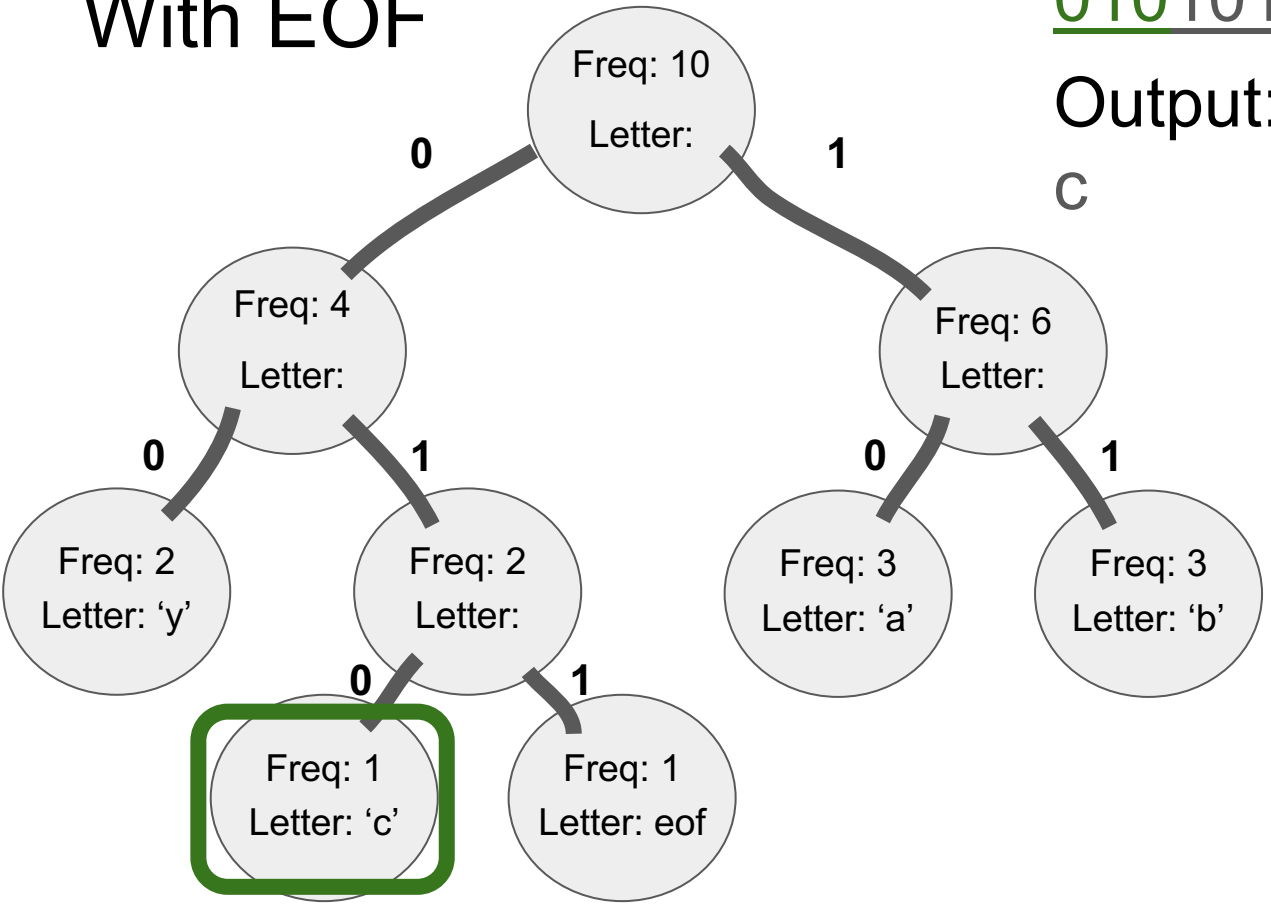
Making a HuffmanCode With EOF

Binary Bits:

01010111 10110000



Making a HuffmanCode With EOF



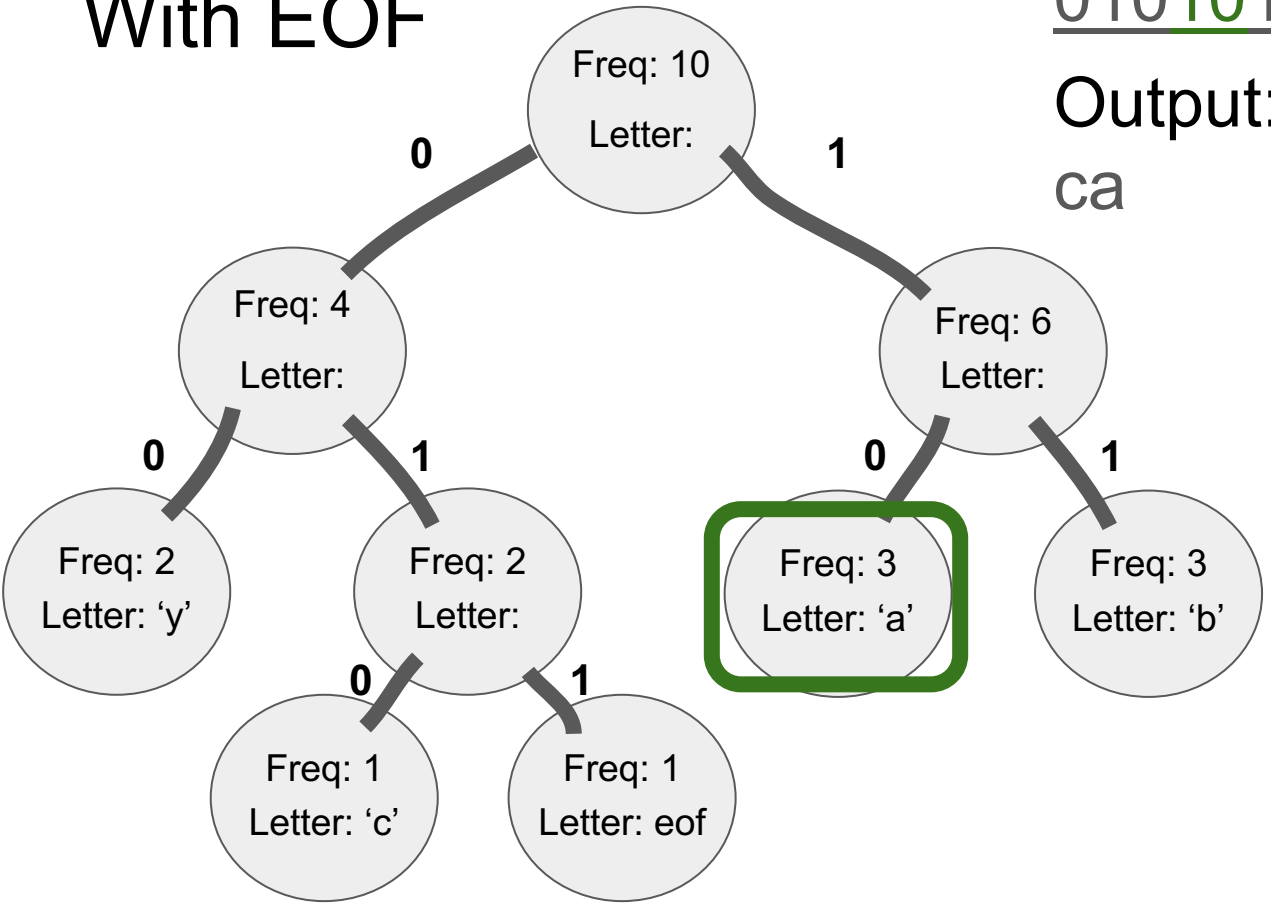
Binary Bits:

010 10111 10110000

Output:

c

Making a HuffmanCode With EOF



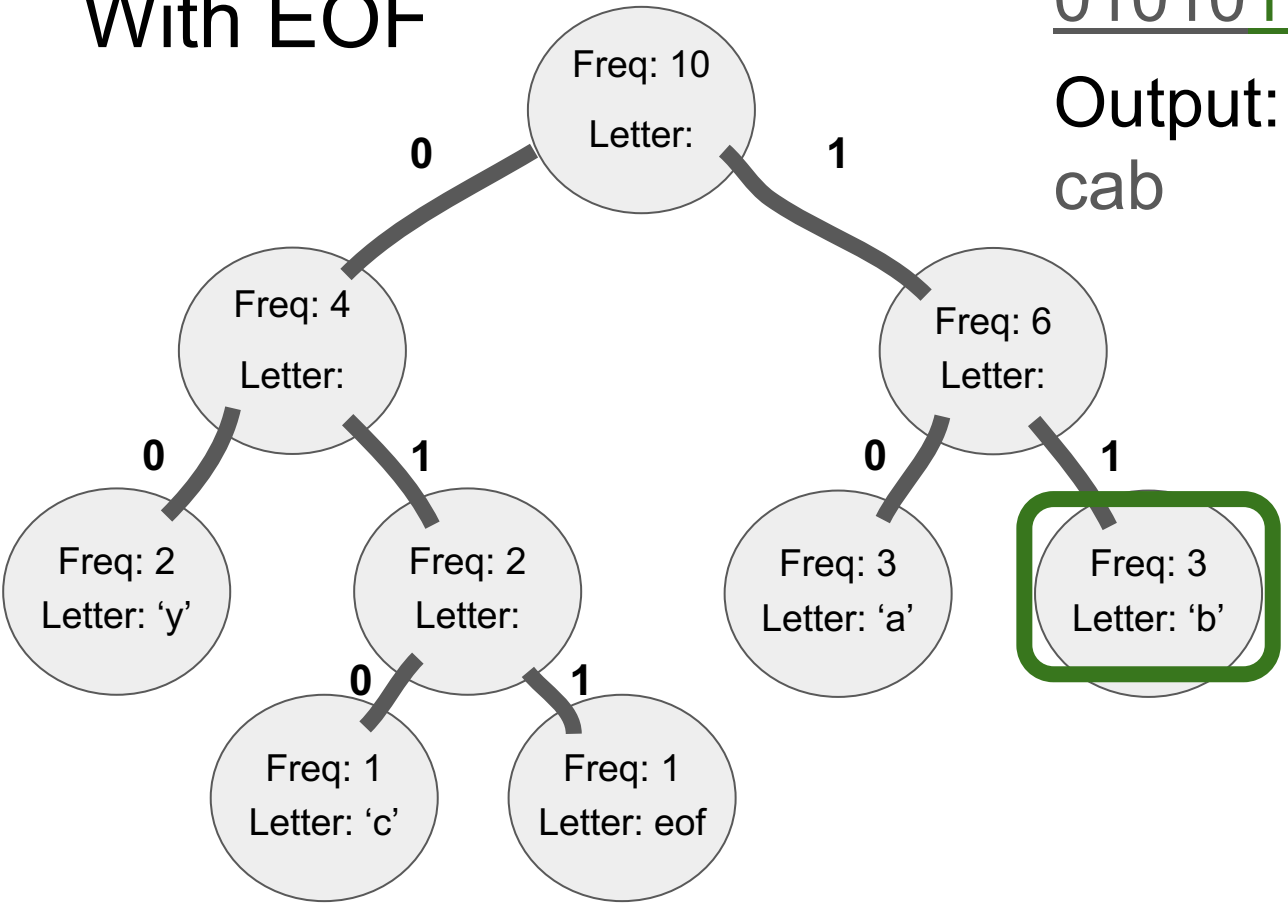
Binary Bits:

01010111 10110000

Output:

ca

Making a HuffmanCode With EOF



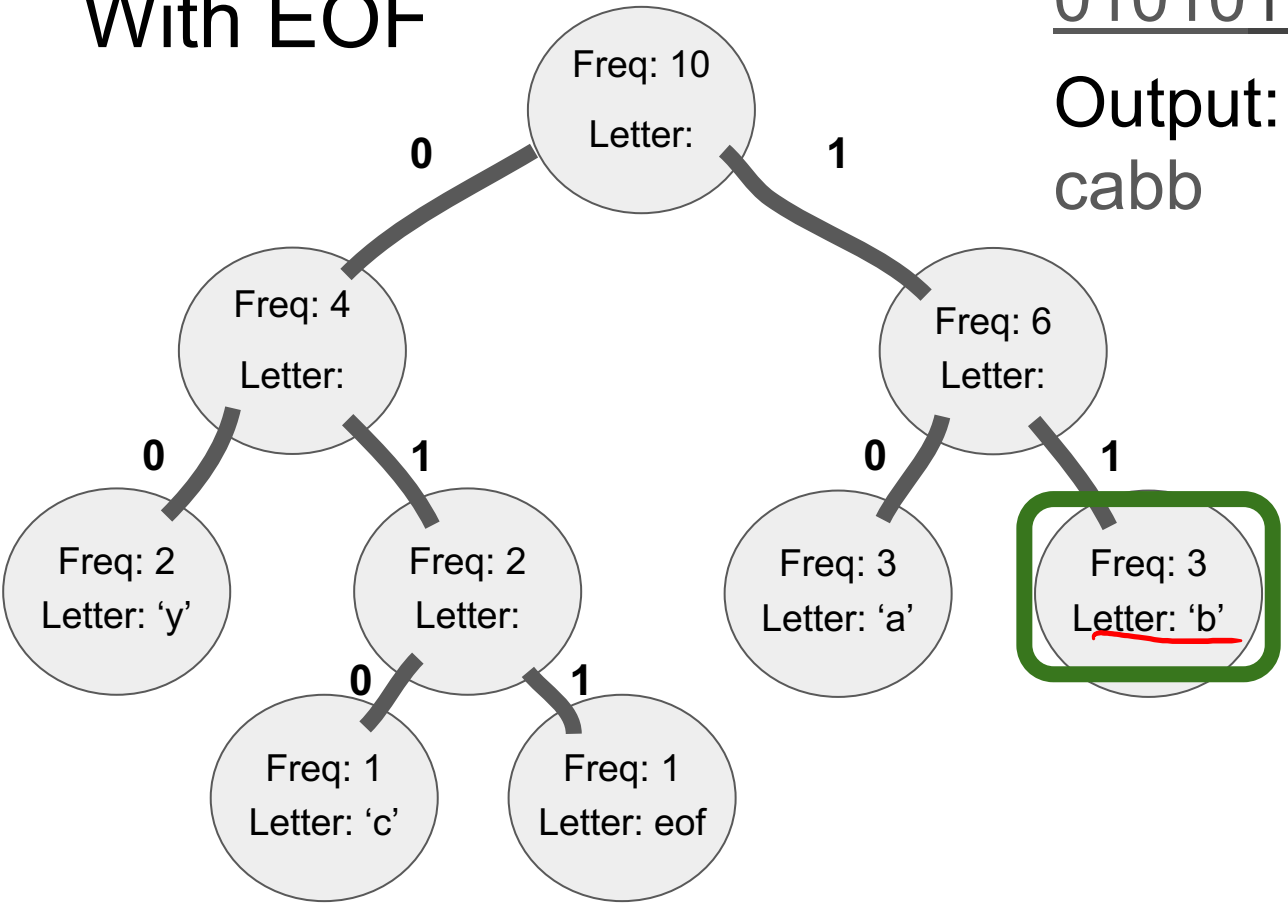
Binary Bits:

01010111 10110000

Output:

cab

Making a HuffmanCode With EOF



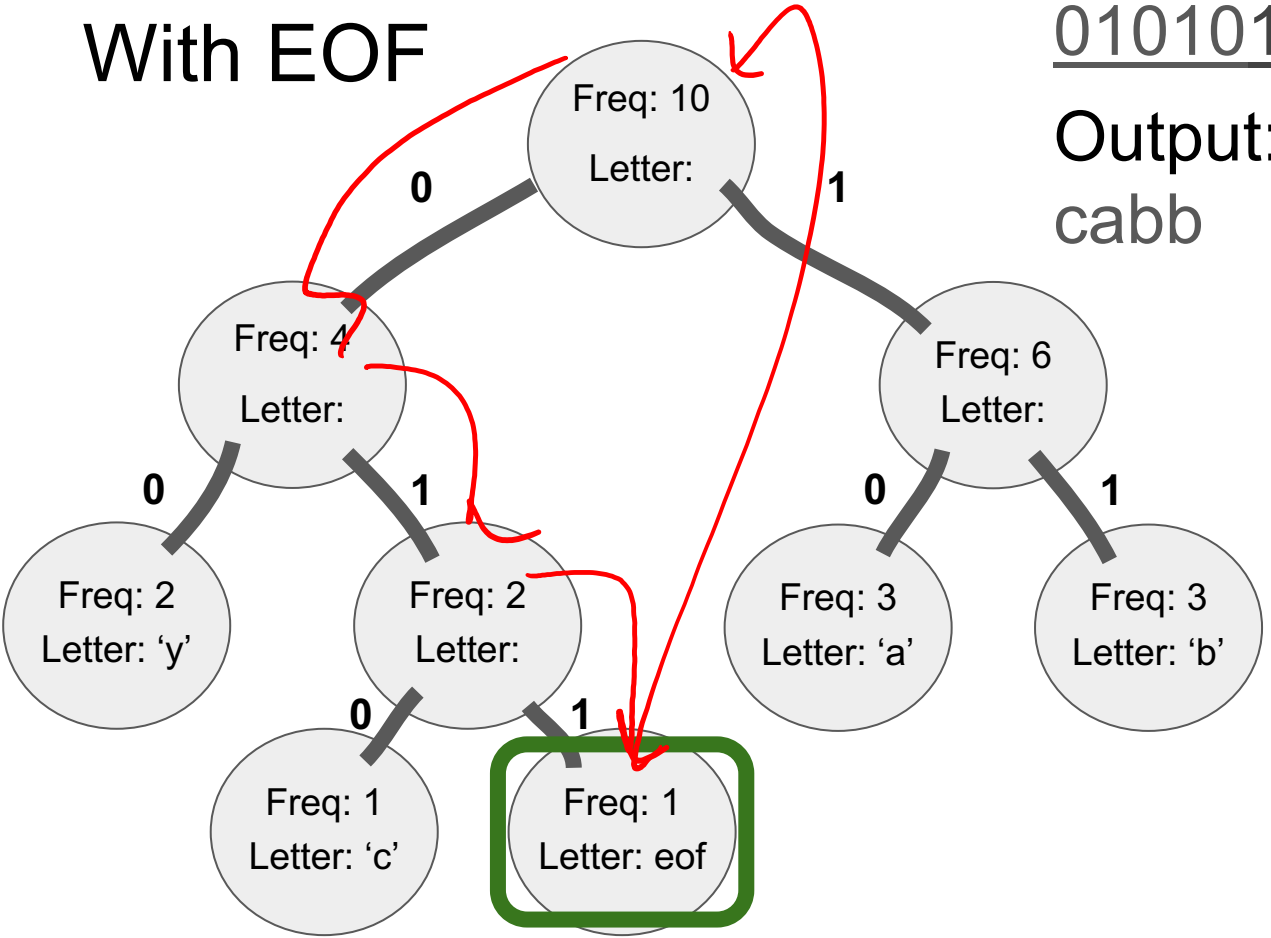
Binary Bits:

01010111 10110000

Output:

cabb

Making a HuffmanCode With EOF



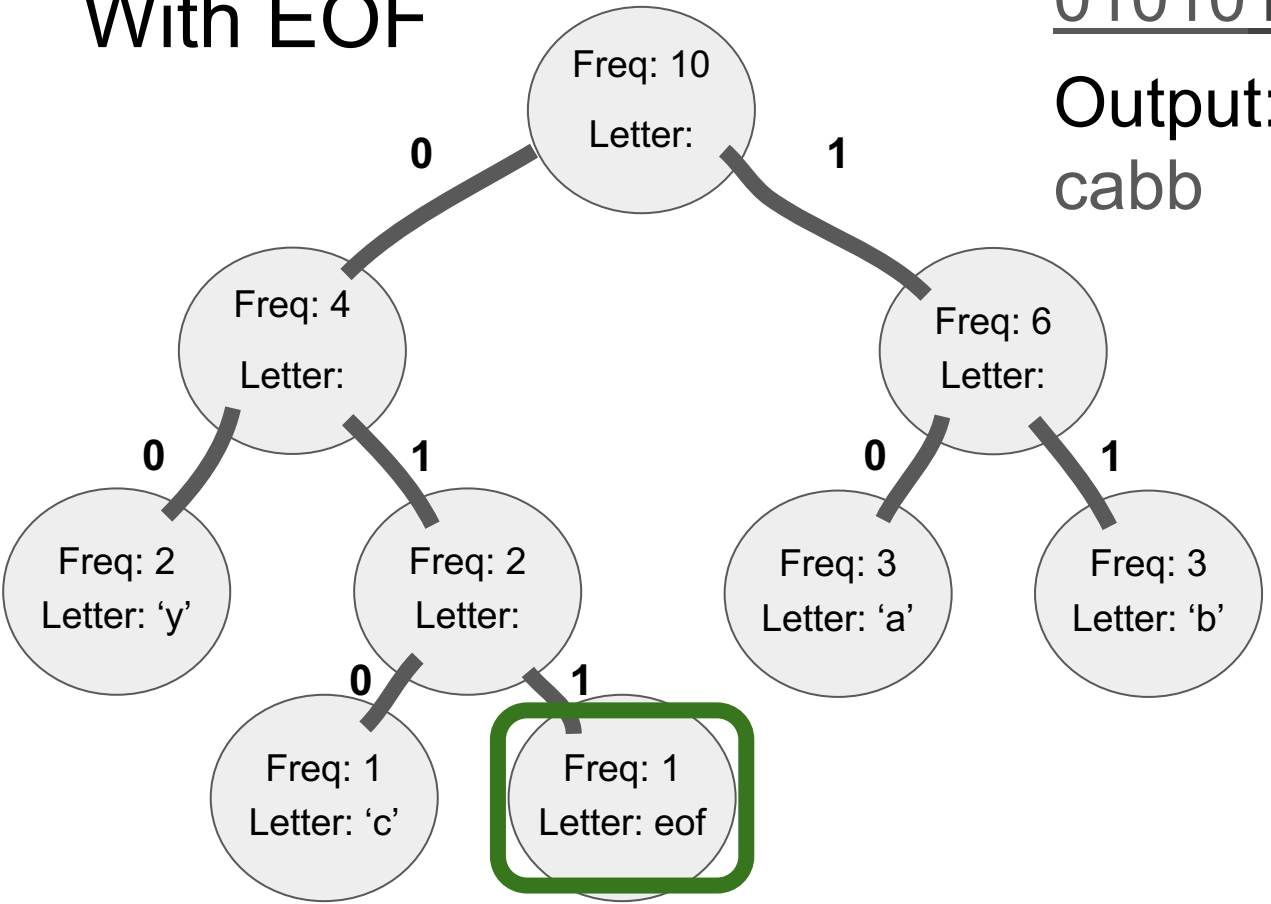
Binary Bits:

01010111 10110000

Output:

cabb

Making a HuffmanCode With EOF



Binary Bits:

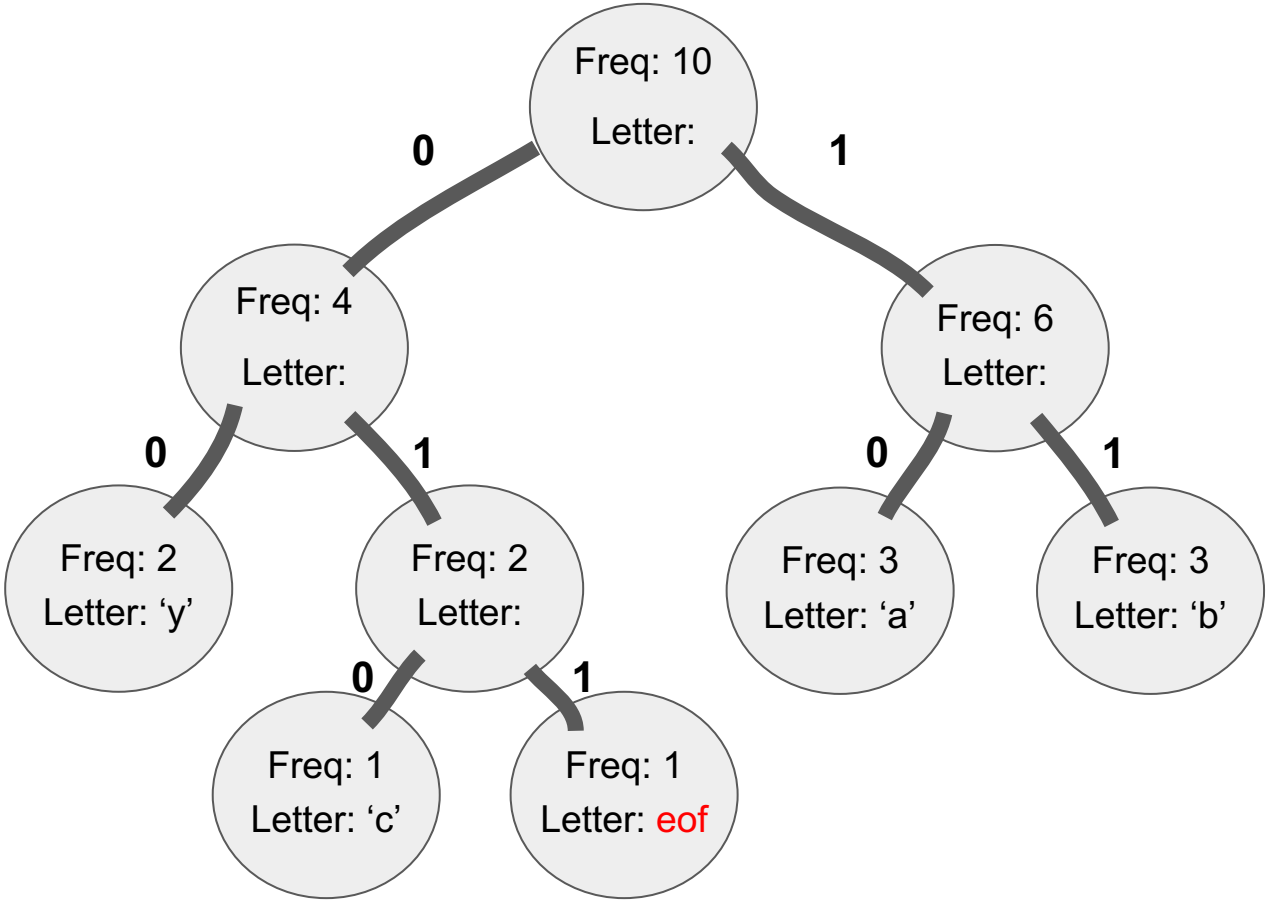
01010111 10110000

Output:

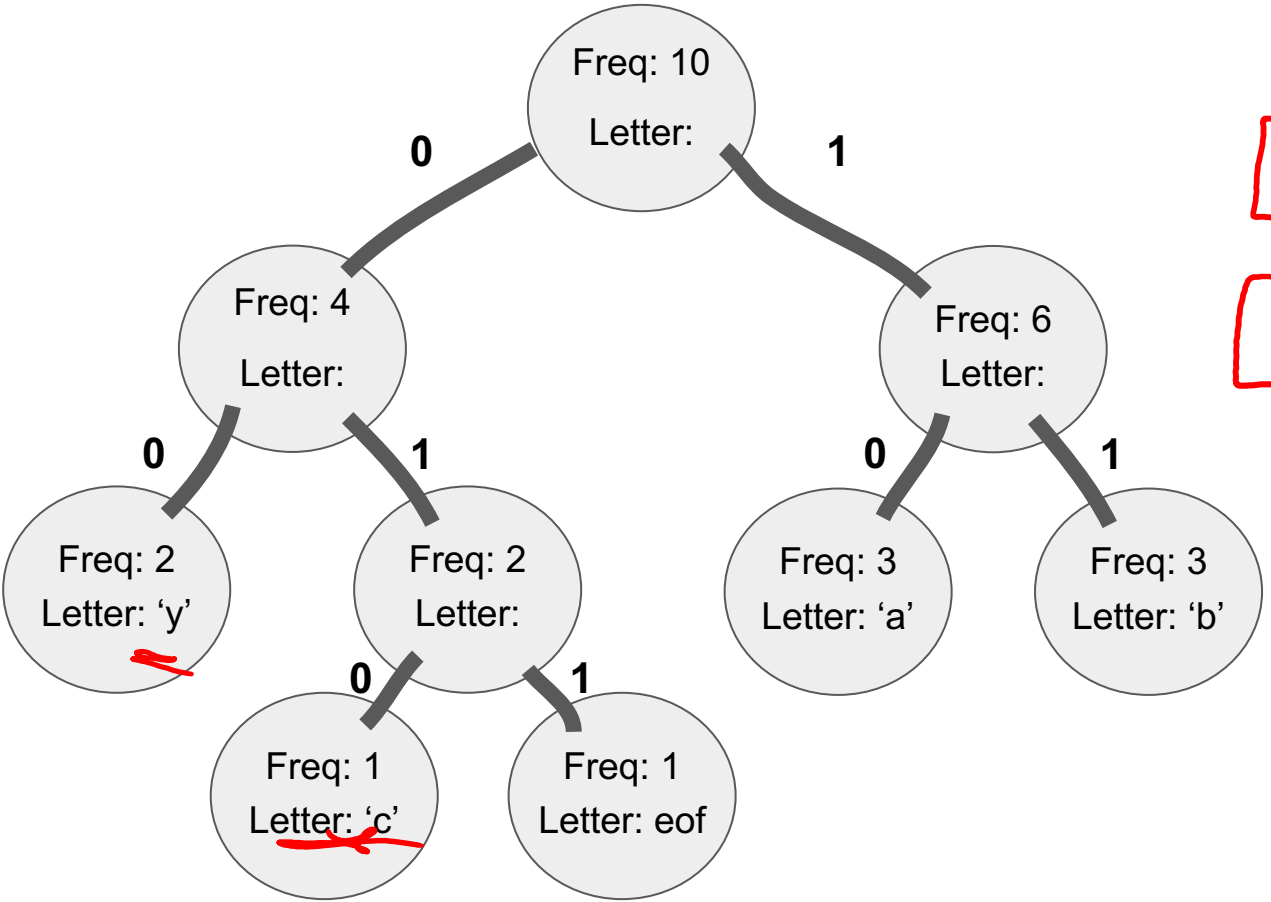
cabb

Extra 0’s don’t matter because we stop at eof

Saving HuffmanCode



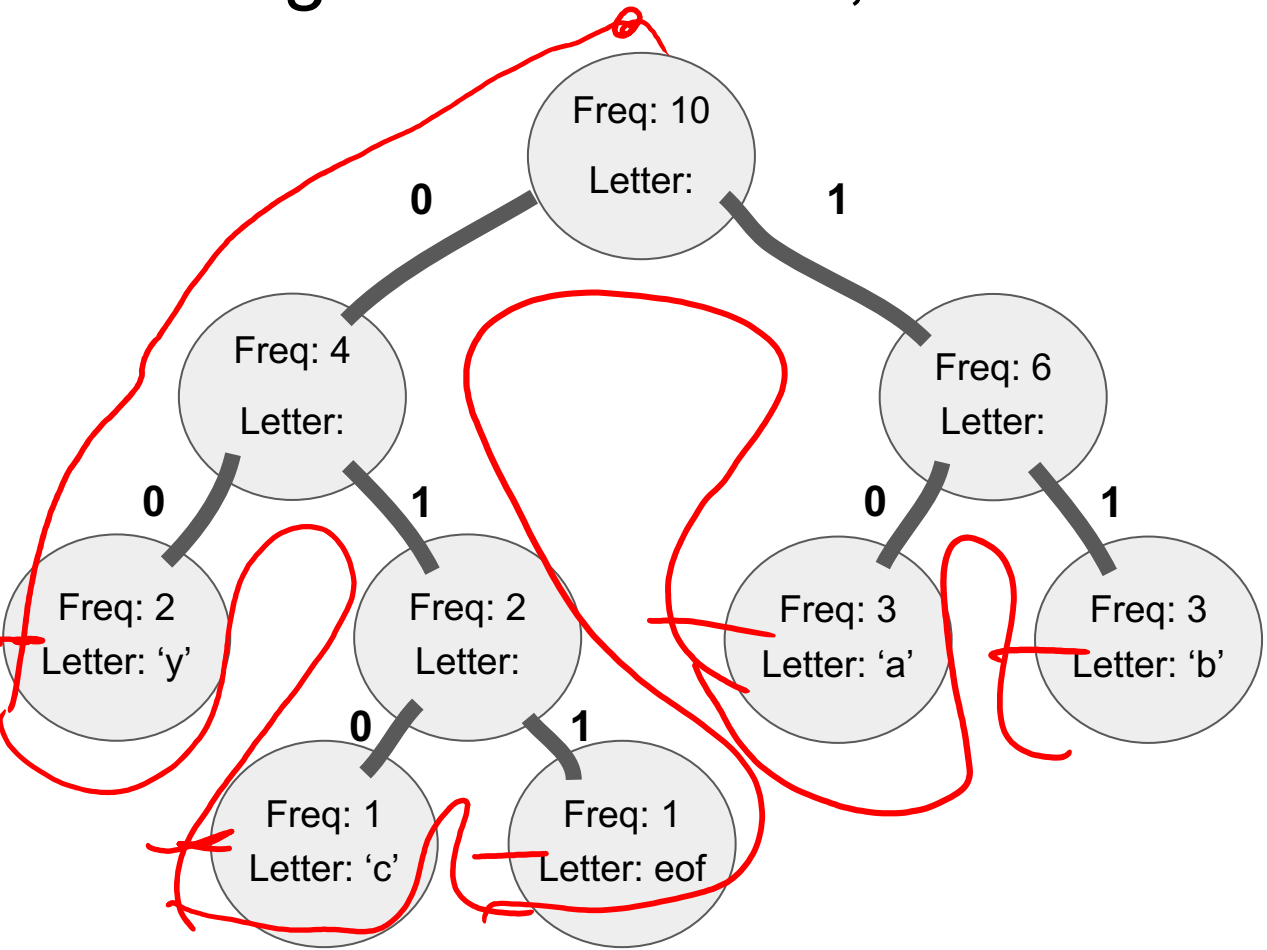
Saving HuffmanCode, .code file



taxi.code

121
00
99
010
255
011
97
10
98
11

Saving HuffmanCode, .code file



taxi.code

121 (y)
00
99 (c)
010
255 (eof)
011
97 (a)
10
98 (b)
11