CSCI 232:
Data Structures and Algorithms

Huffman Coding

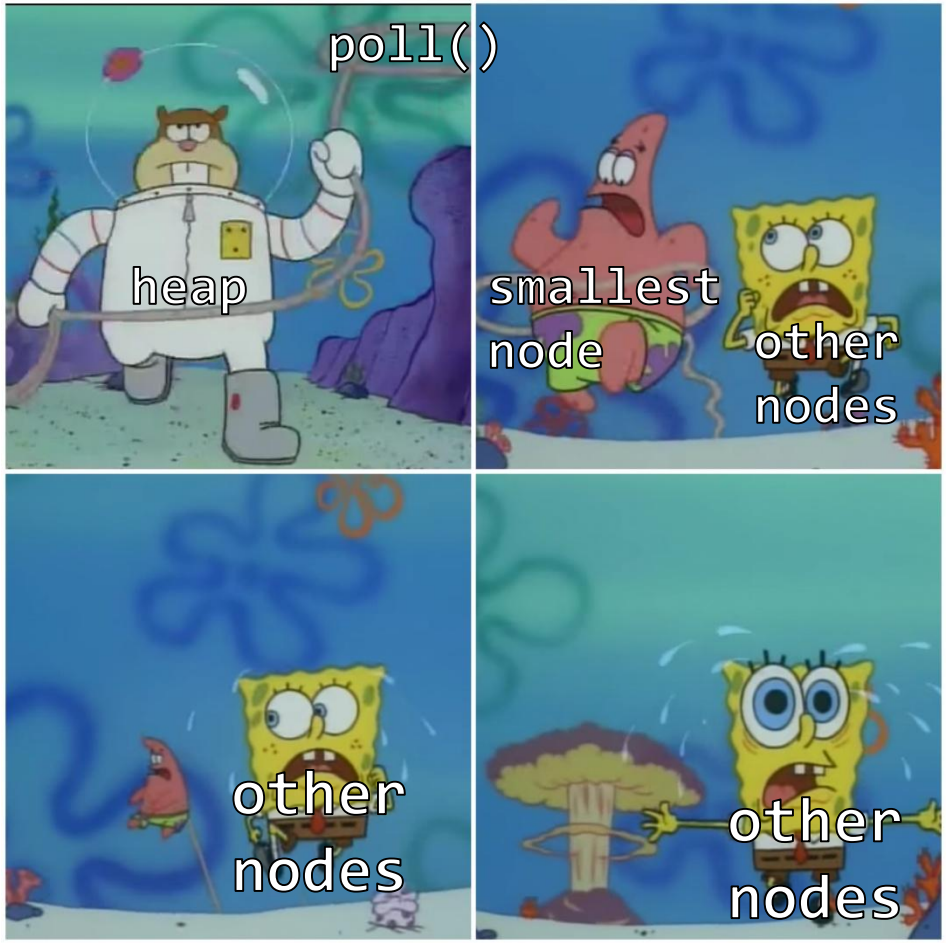
Reese Pearsall

Summer 2025

Announcements

Program 2 due **Wednesday** at 11:59 PM

Quiz 2 on Thursday (No lecture)



Strings are **encoded** using a specific format

Convert characters to zeros and ones (binary)

Example: **UTF**

Character	UTF Encoding
a	01100001
b	01100010
c	01100011
d	01100100
e	01100101
…	….

Each character has an **8-bit** binary representation

String: “hello”

UTF Encoding:
01101000 01100101 01101100 01101100 01101111

Strings are **encoded** using a specific format

Convert characters to zeros and ones (binary)

Example: **UTF**

Character	UTF Encoding
a	01100001
b	01100010
c	01100011
d	01100100
e	01100101
…	….

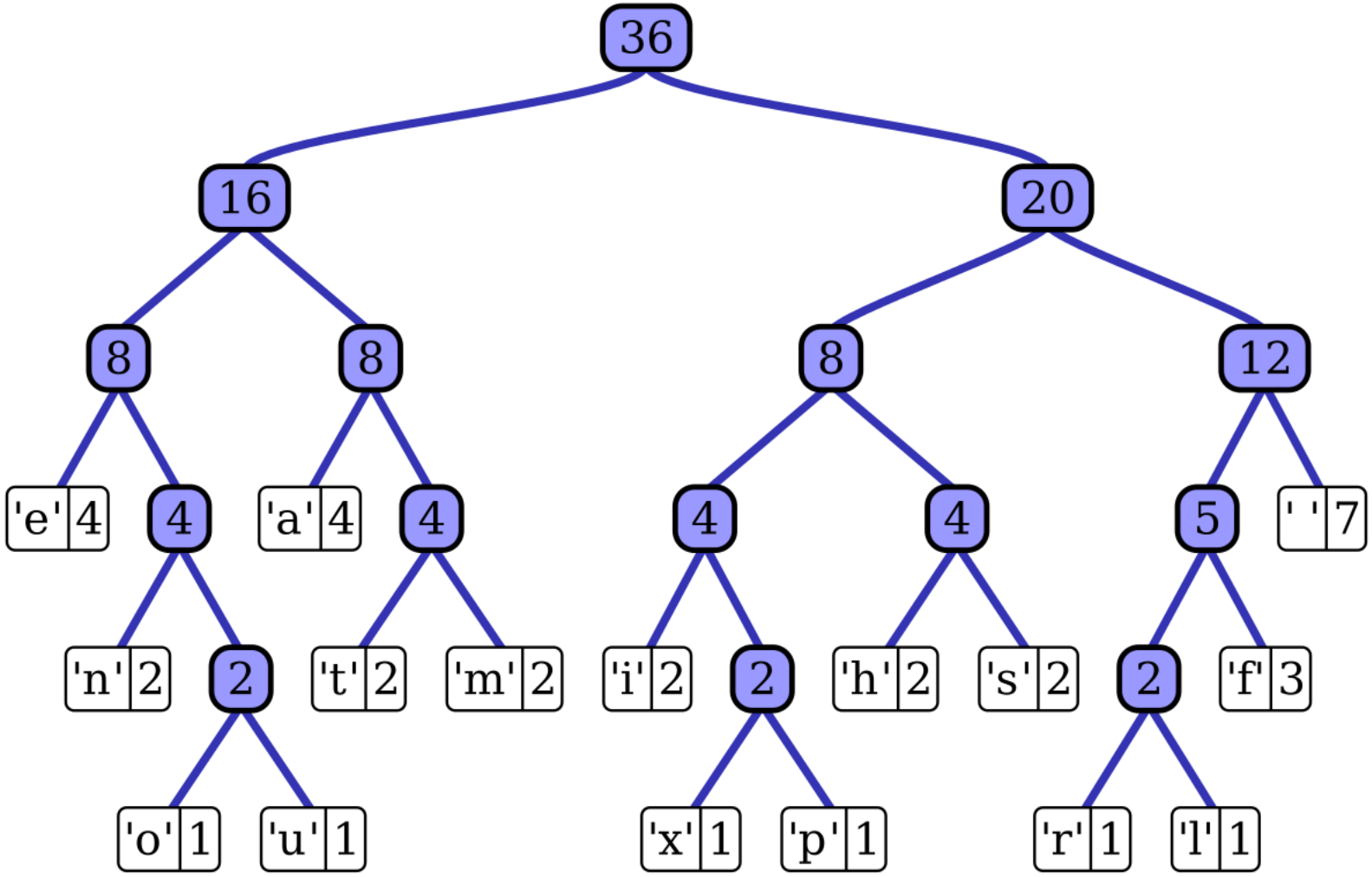
Each character has an **8-bit** binary representation

String: “hello”

UTF Encoding:

01101000 01100101 01101100 01101100 01101111

Huffman Coding is a way to encode a string using a binary tree



Internal Nodes are the sum of the children's frequencies

Leaf nodes are characters with their frequency

When the Huffman tree is built, we can extract a binary encoding for each character

Huffman Coding

Step 1: Generate the frequencies of each character, and sort them from least to greatest

Huffman Coding

Step 1: Generate the frequencies of each character, and sort them from least to greatest

"hello world!"

char	freq
	1
!	1
d	1
e	1
h	1
r	1
w	1
o	2
l	3

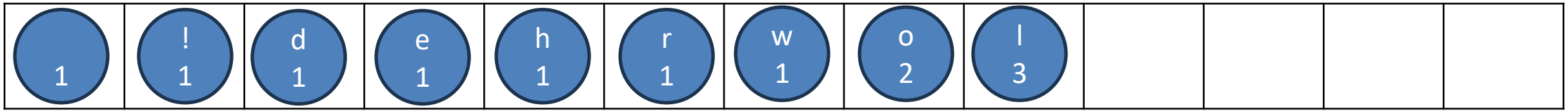
Step 2: Insert them into a PriorityQueue

Huffman Coding

Step 3. Build subtrees, and merge trees into Huffman Tree

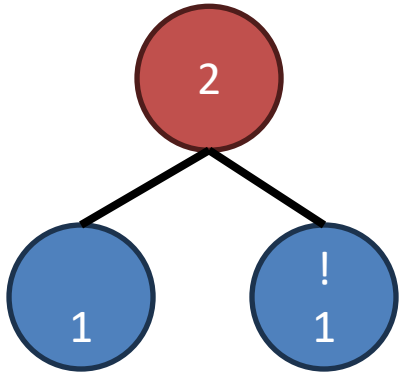
Create an internal node with the two nodes at front of queue,
Place new internal node in queue

PriorityQueue<HuffmanNode>

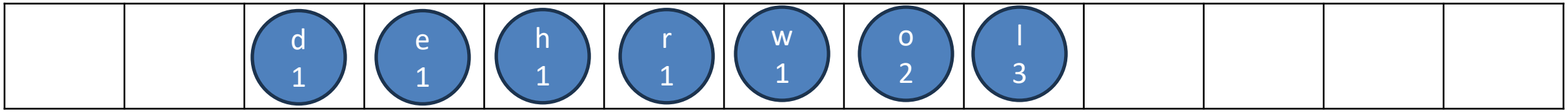


Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue

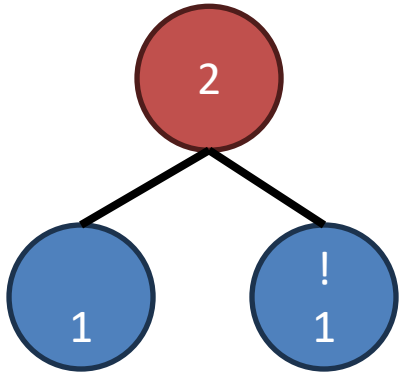


PriorityQueue<HuffmanNode>

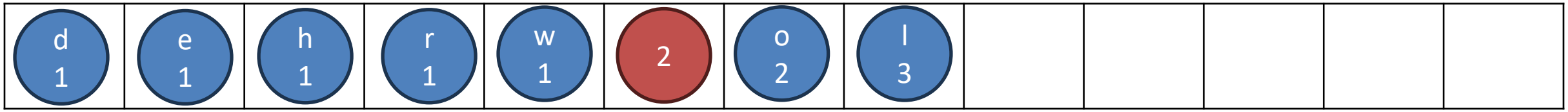


Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue



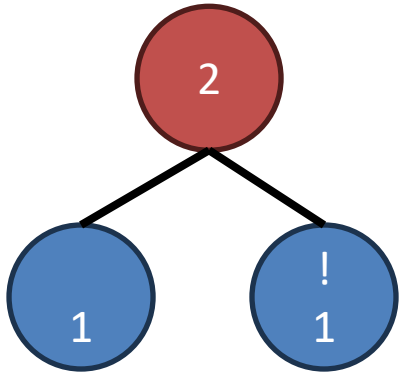
PriorityQueue<HuffmanNode>



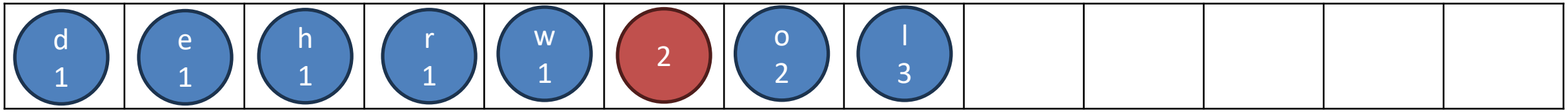
Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue

Repeat until all nodes are inserted!

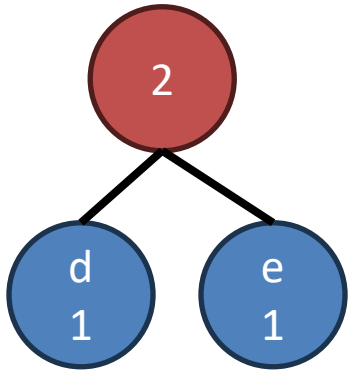


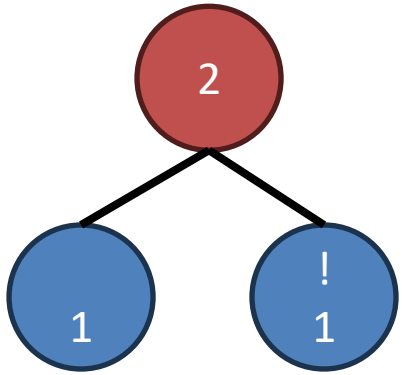
PriorityQueue<HuffmanNode>



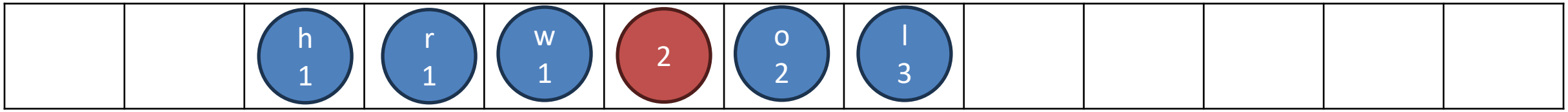
Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue



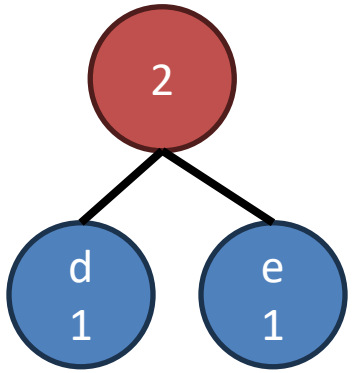


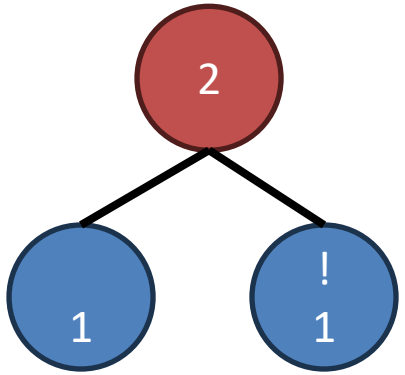
PriorityQueue<HuffmanNode>



Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue





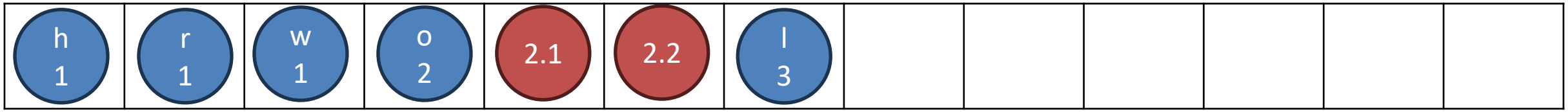
PriorityQueue<HuffmanNode>

Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue

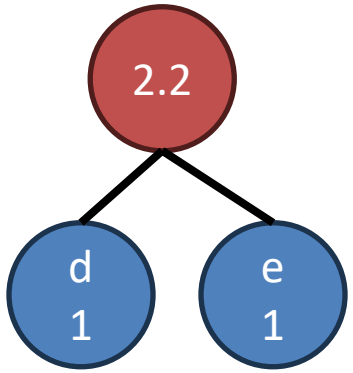
These nodes have frequencies of **two**, but I am adding a way to distinguish them in the queue (.1, .2, .3)

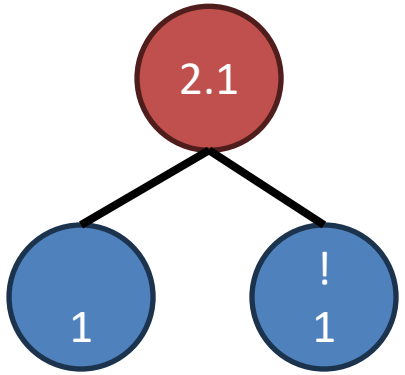
PriorityQueue<HuffmanNode>



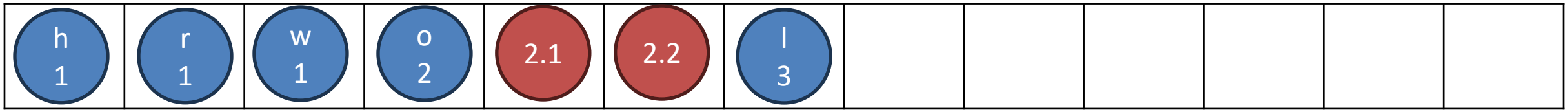
Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue



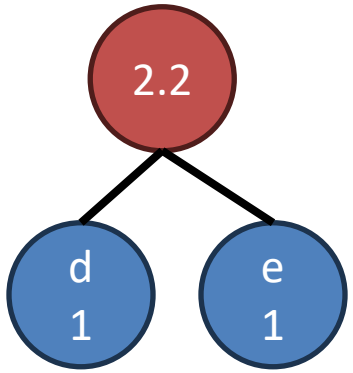


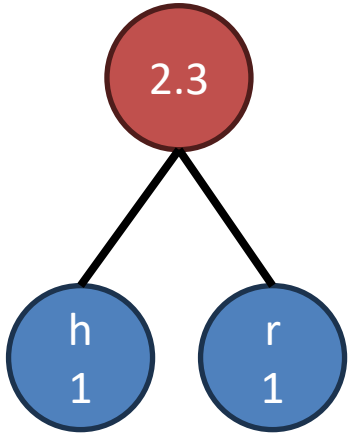
PriorityQueue<HuffmanNode>

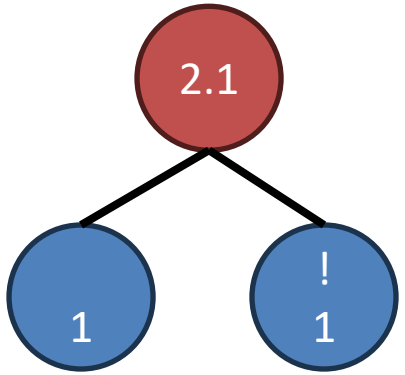


Step 3. Build subtrees, and merge trees into Huffman Tree

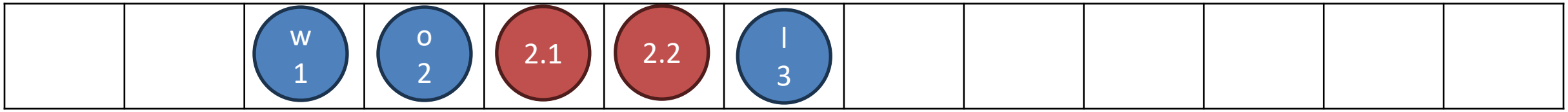
Create an internal node with the two nodes at front of queue,
Place new internal node in queue





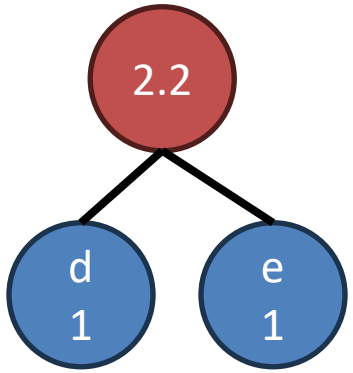


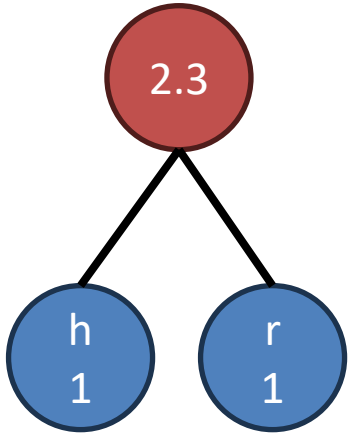
PriorityQueue<HuffmanNode>

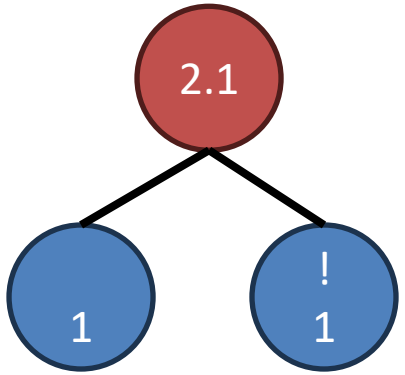


Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue





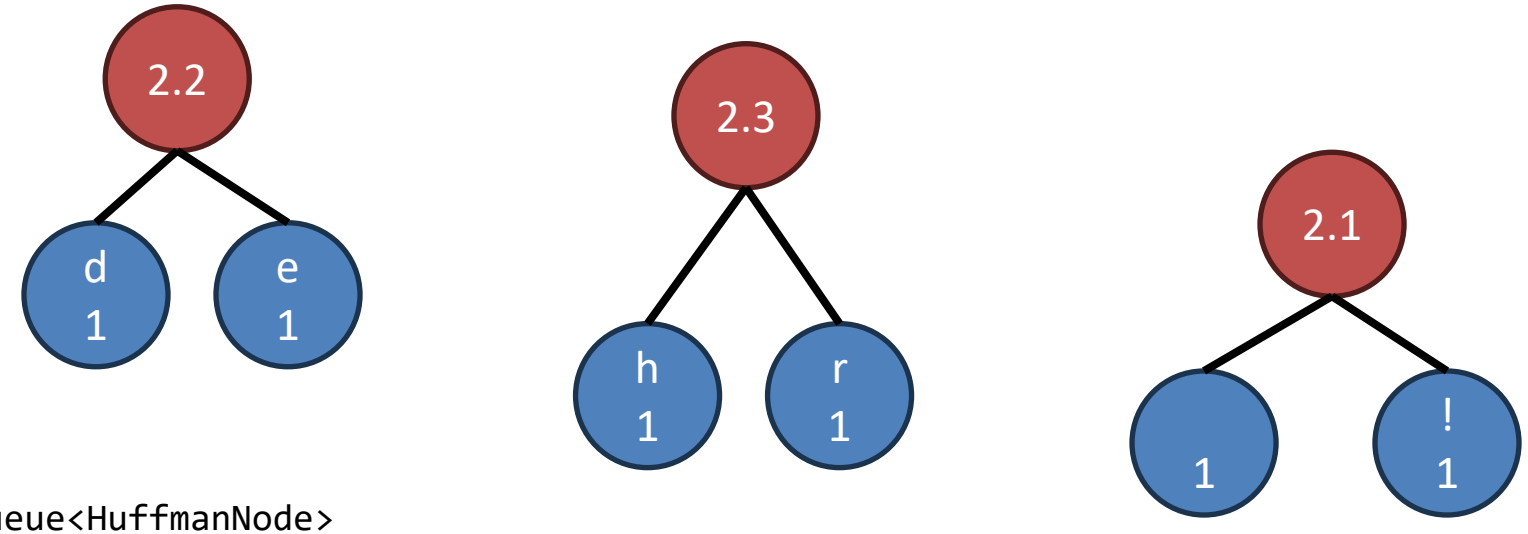


`PriorityQueue<HuffmanNode>`

Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue

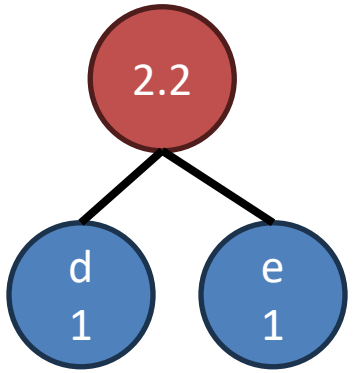
Because it is a priority queue, it is not necessary FIFO…

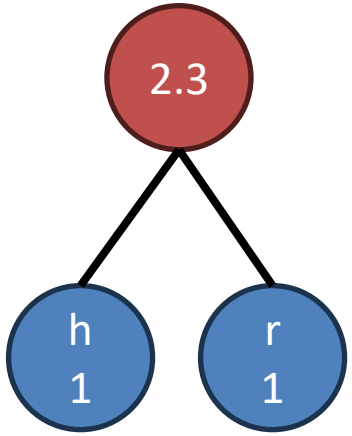


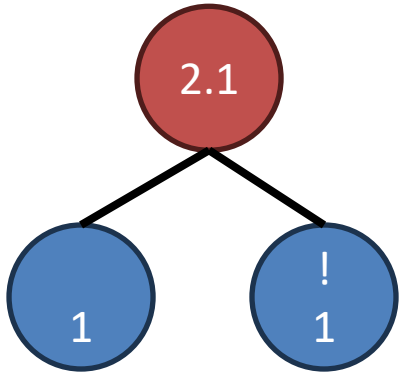
PriorityQueue<HuffmanNode>

Step 3. Build subtrees, and merge trees into Huffman Tree

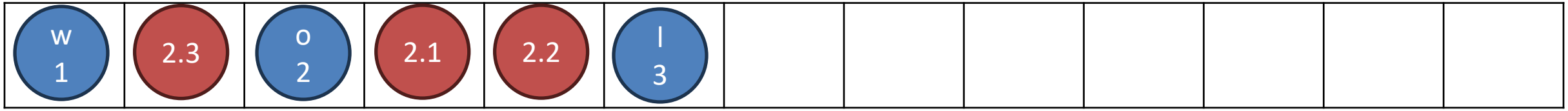
Create an internal node with the two nodes at front of queue,
Place new internal node in queue





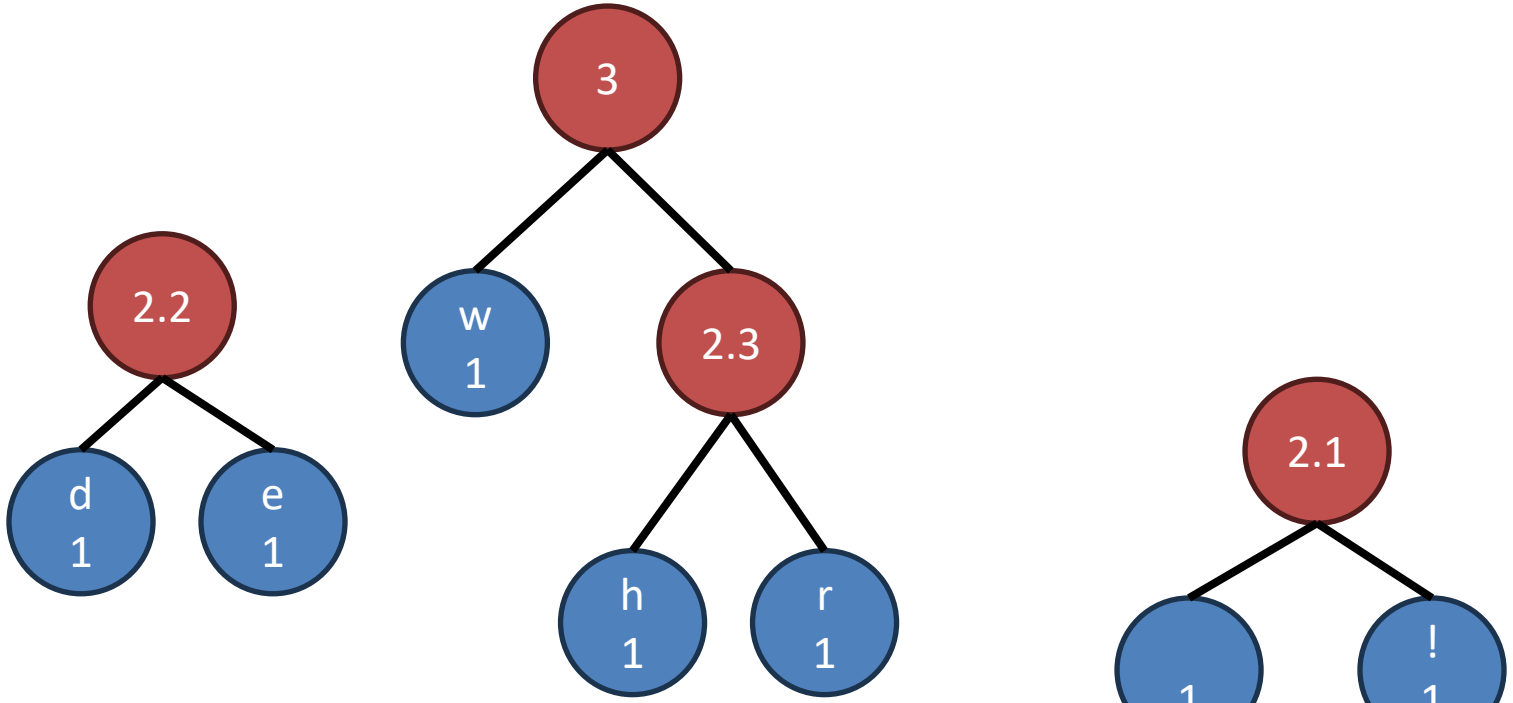


PriorityQueue<HuffmanNode>



Step 3. Build subtrees, and merge trees into Huffman Tree

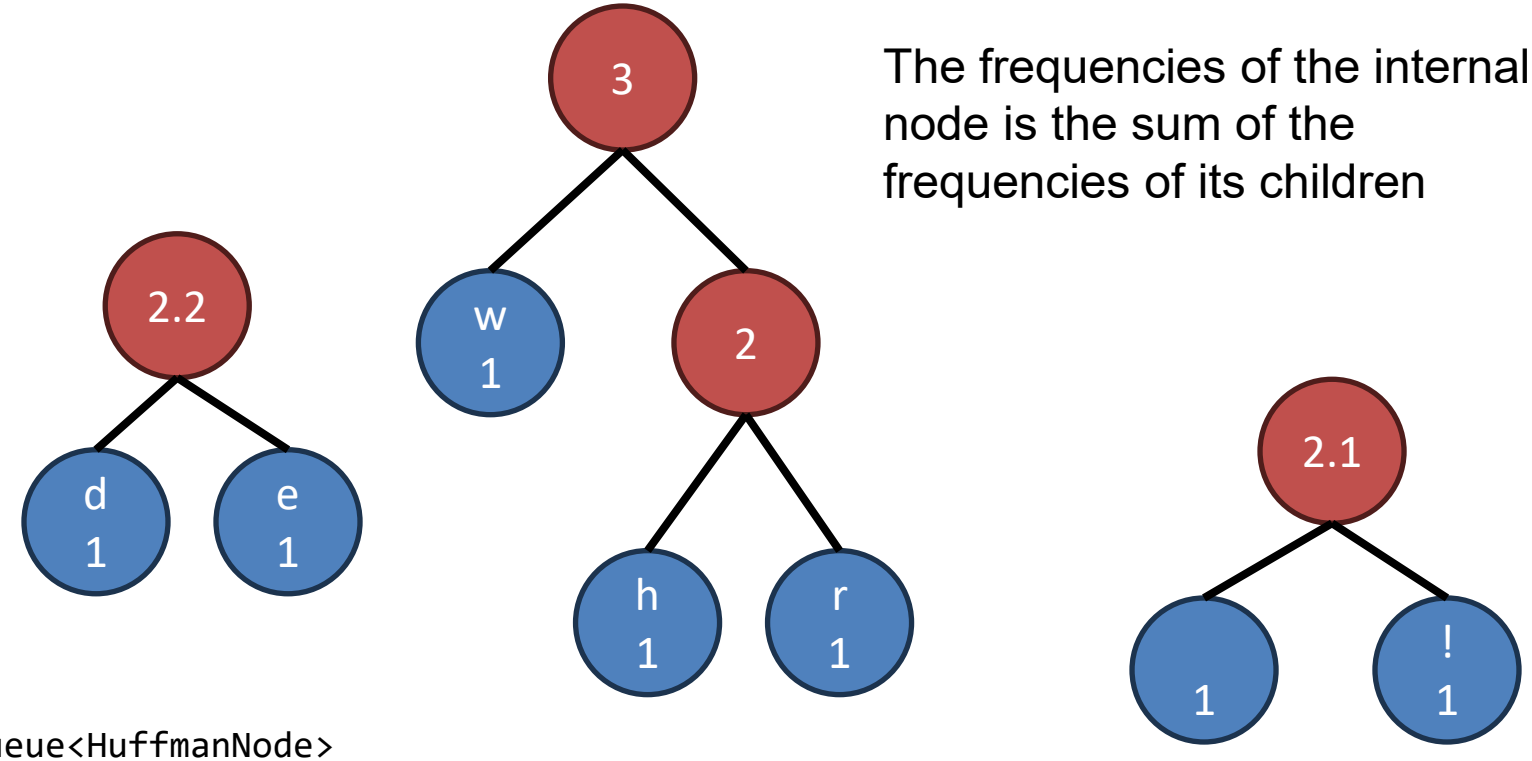
Create an internal node with the two nodes at front of queue,
Place new internal node in queue



PriorityQueue<HuffmanNode>

Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue



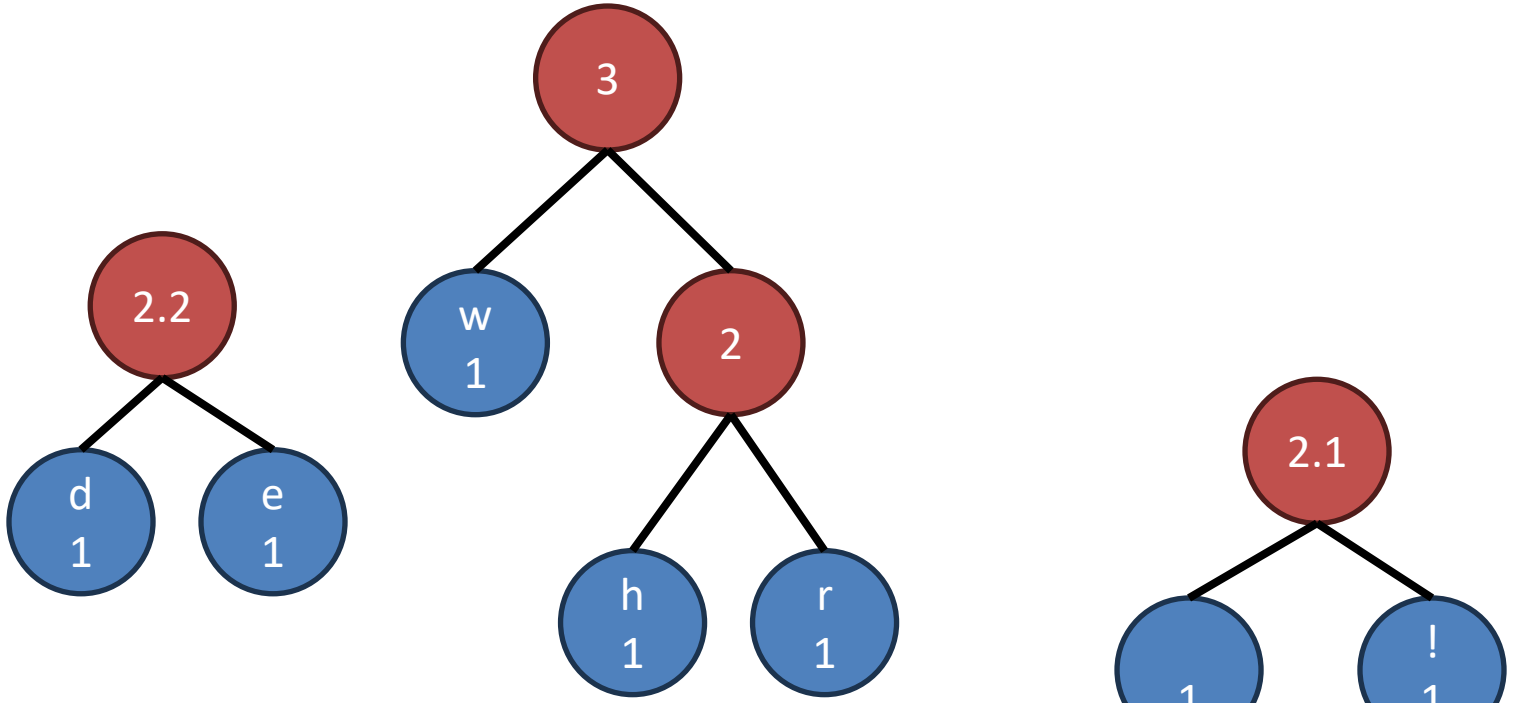
The frequencies of the internal node is the sum of the frequencies of its children

PriorityQueue<HuffmanNode>

Huffman Coding

Step 3. Build subtrees, and merge trees into Huffman Tree

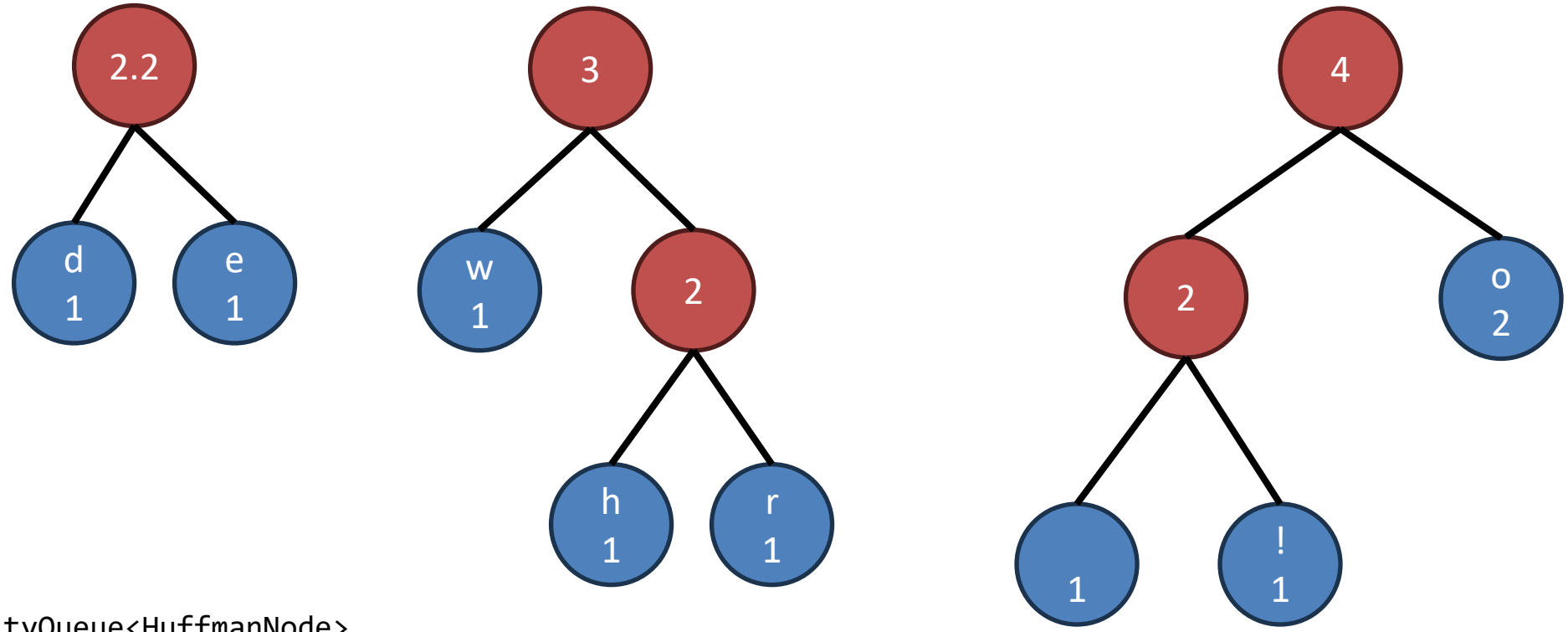
Create an internal node with the two nodes at front of queue,
Place new internal node in queue



PriorityQueue<HuffmanNode>

Step 3. Build subtrees, and merge trees into Huffman Tree

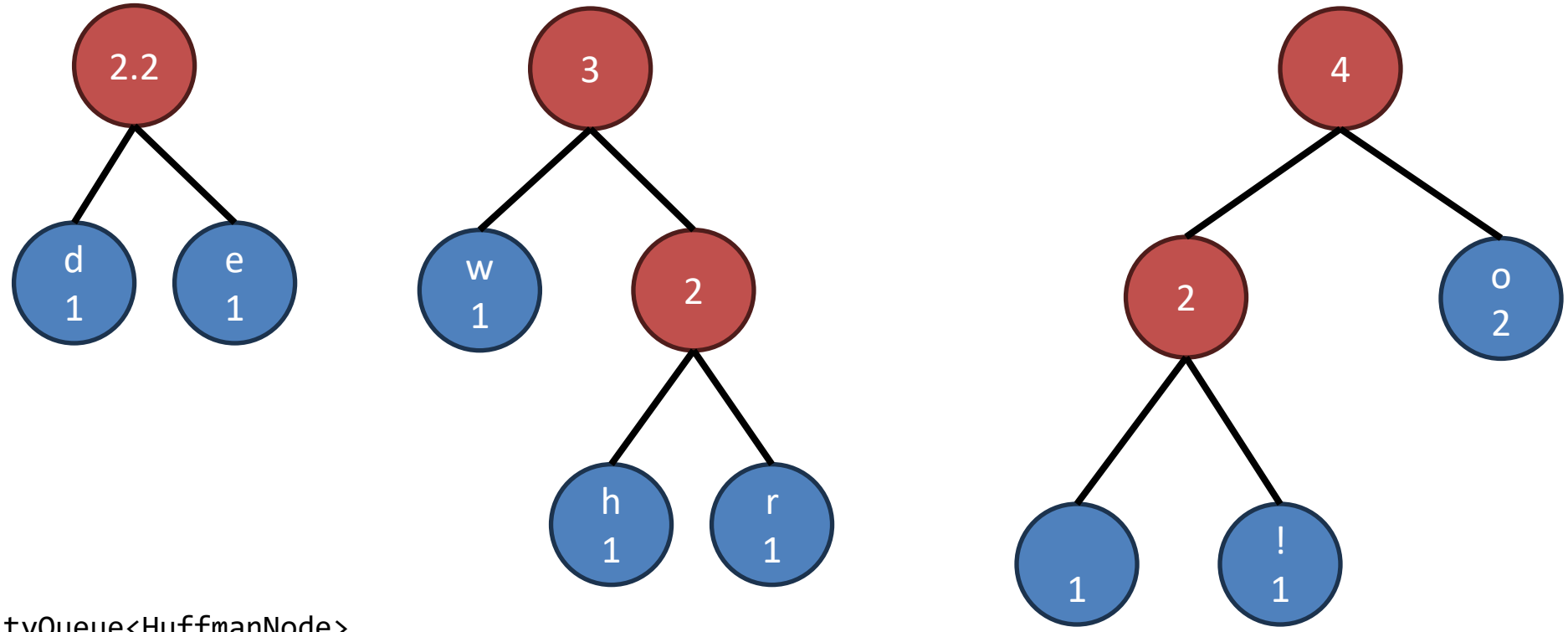
Create an internal node with the two nodes at front of queue,
Place new internal node in queue



PriorityQueue<HuffmanNode>

Step 3. Build subtrees, and merge trees into Huffman Tree

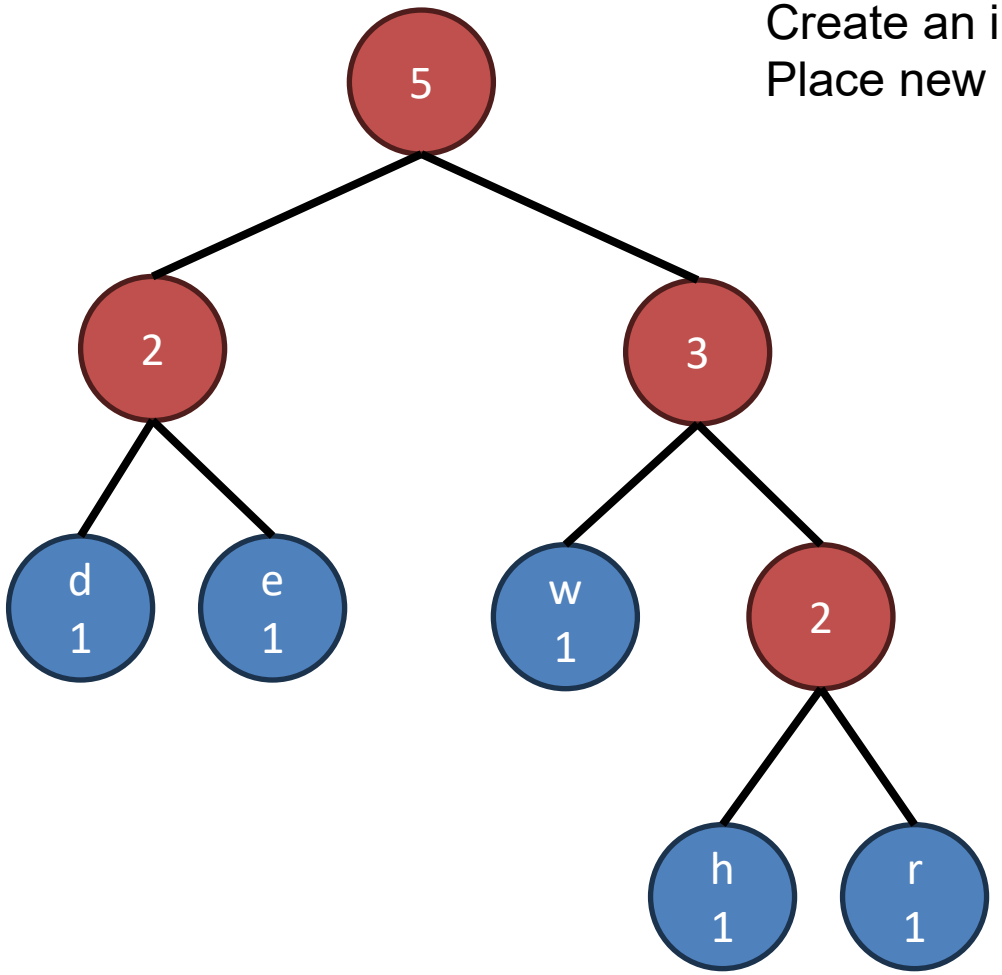
Create an internal node with the two nodes at front of queue,
Place new internal node in queue

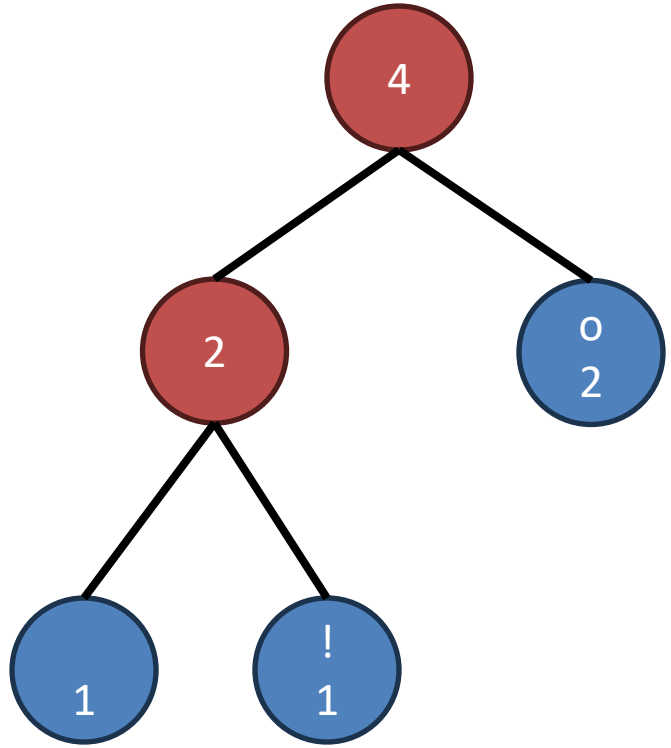


PriorityQueue<HuffmanNode>

Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue

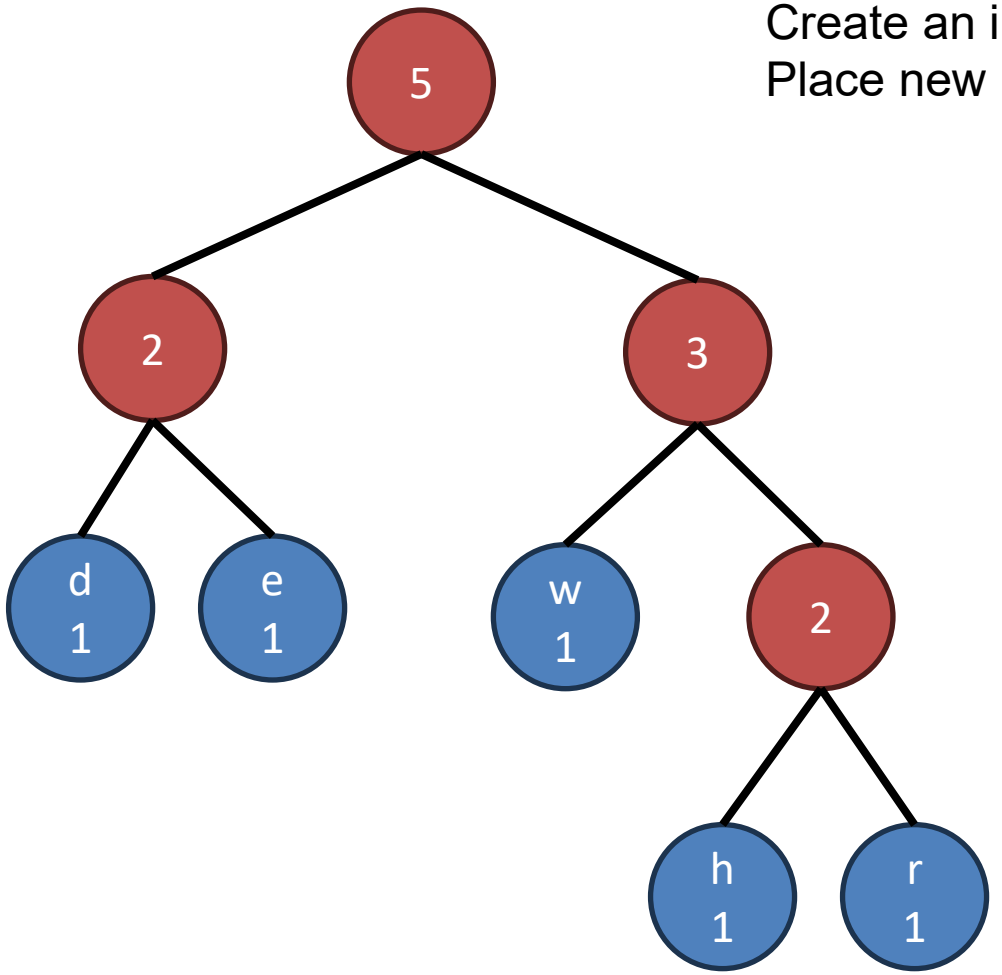


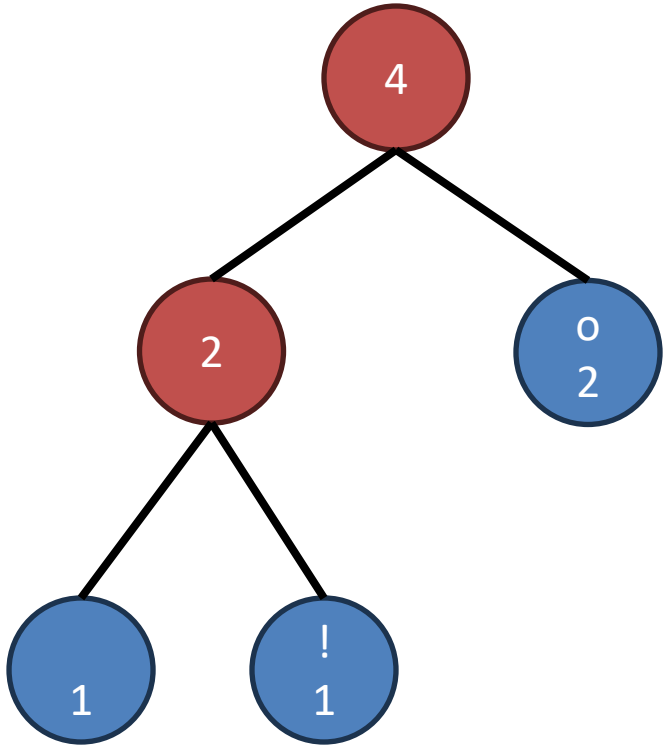


PriorityQueue<HuffmanNode>

Step 3. Build subtrees, and merge trees into Huffman Tree

Create an internal node with the two nodes at front of queue,
Place new internal node in queue

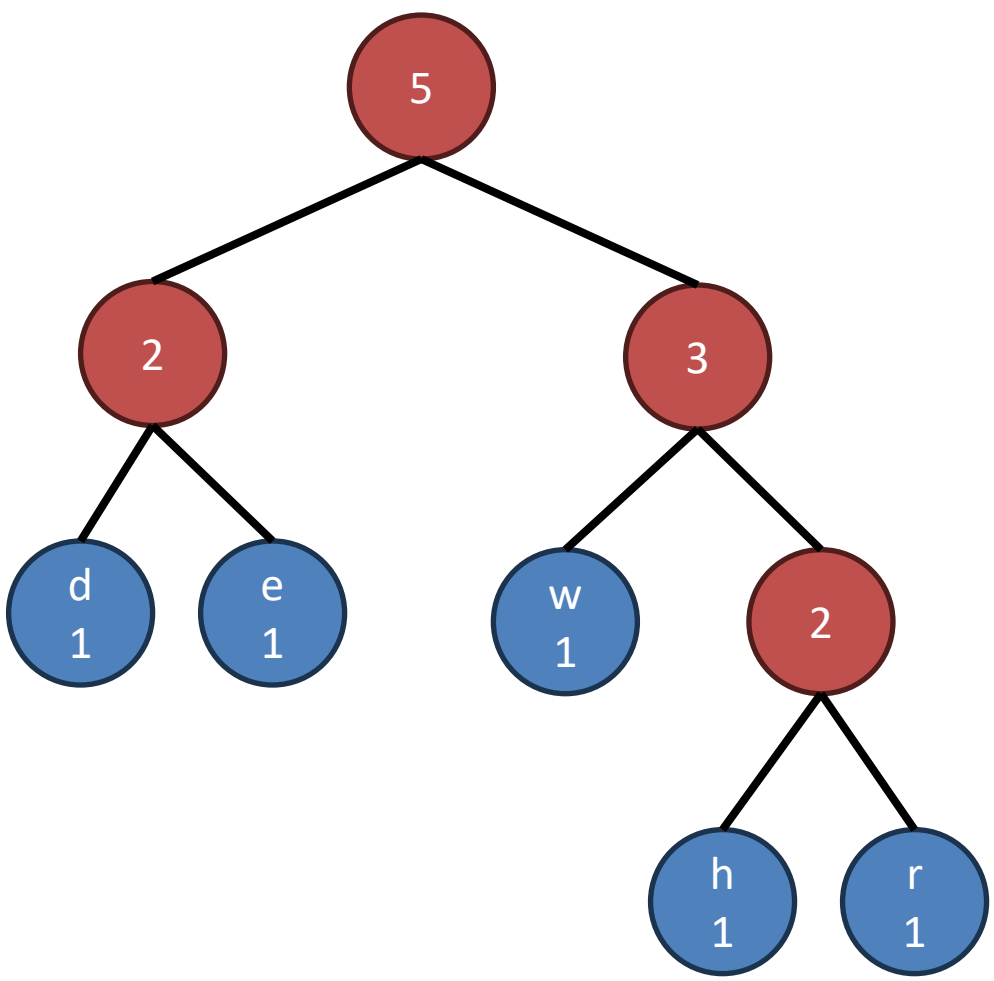


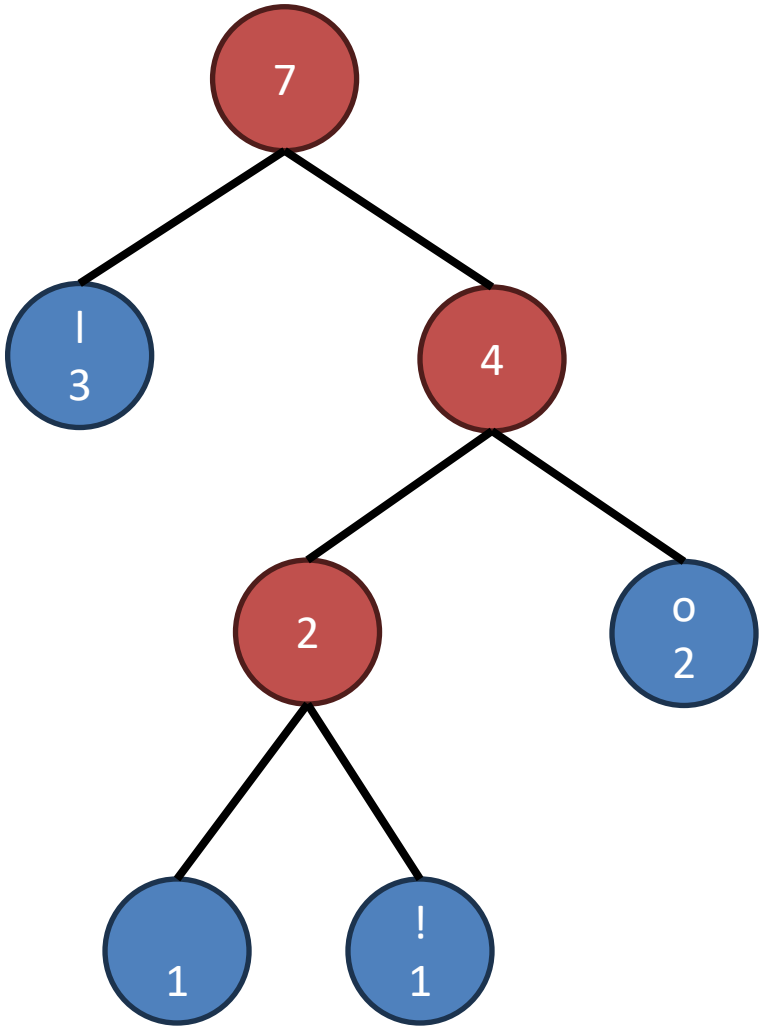


PriorityQueue<HuffmanNode>

Huffman Coding

Step 3. Build subtrees, and merge trees into Huffman Tree

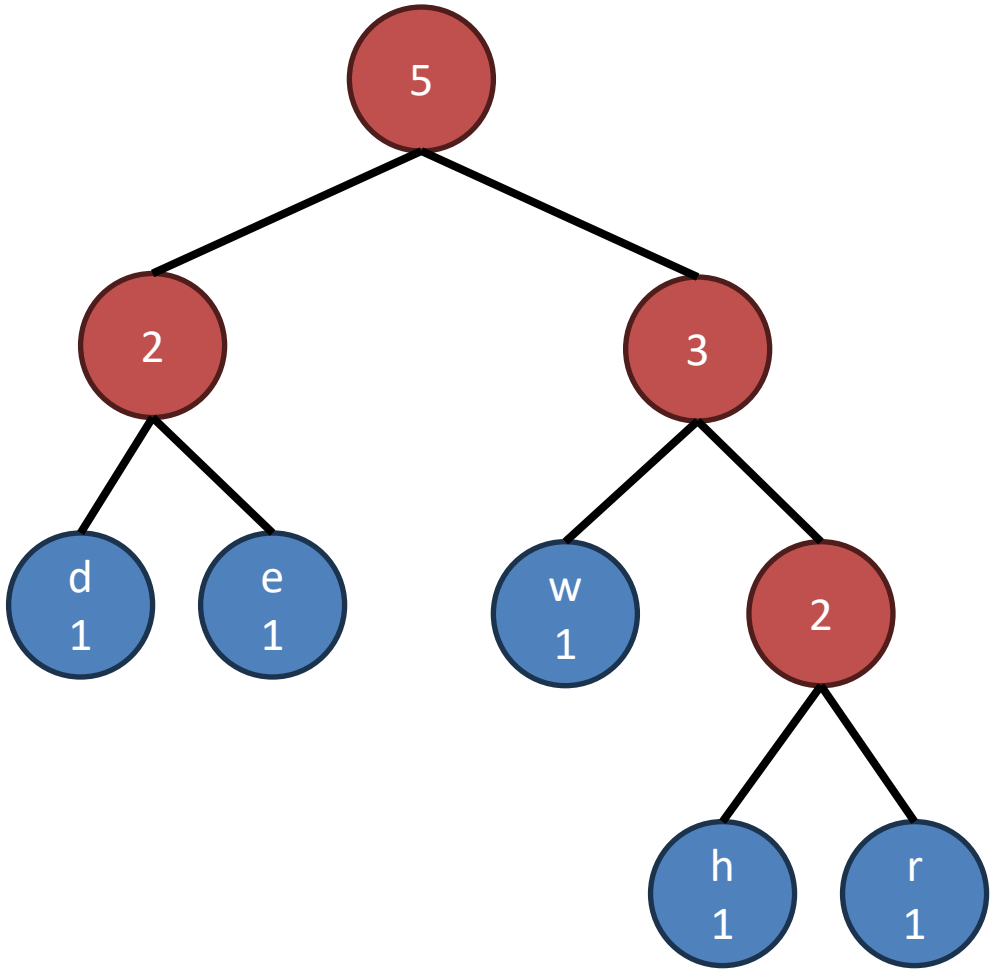


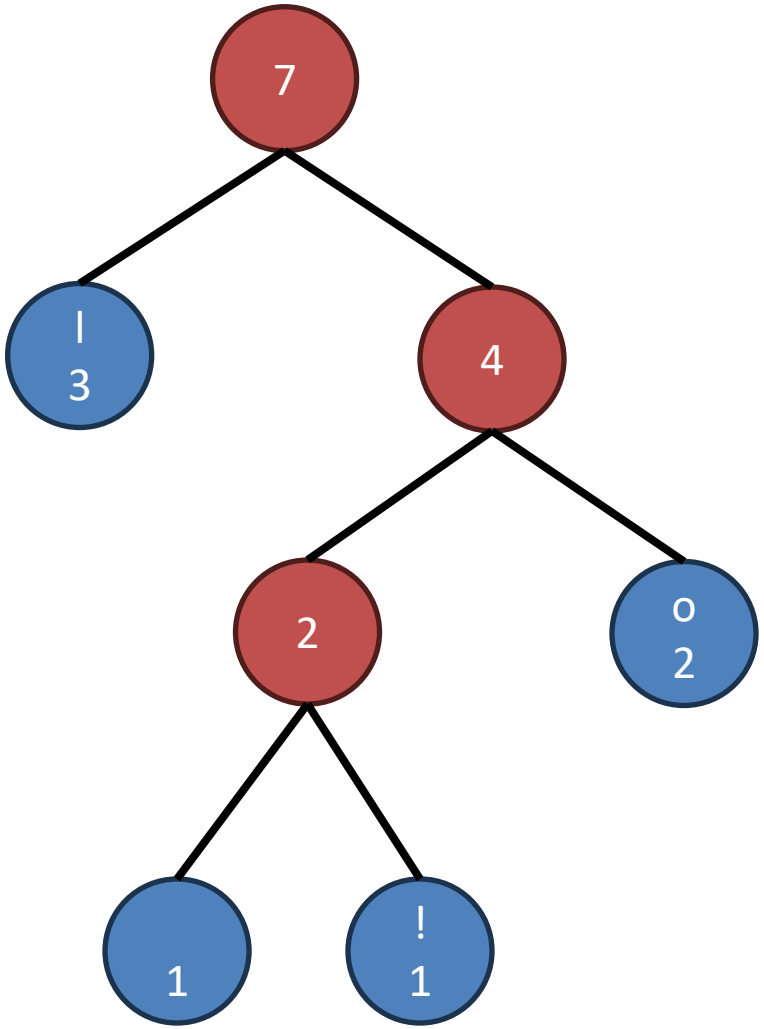


PriorityQueue<HuffmanNode>

Huffman Coding

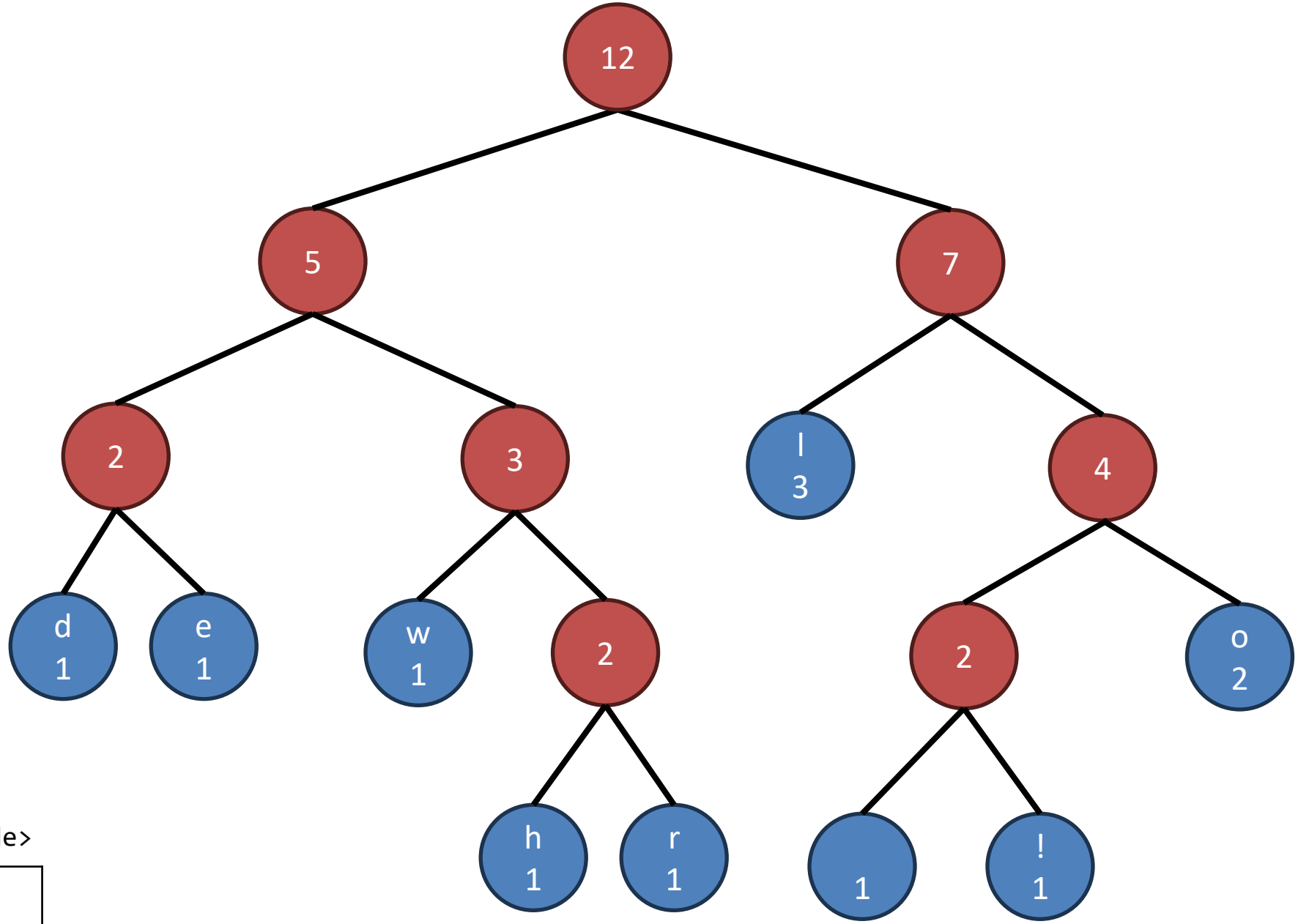
Step 3. Build subtrees, and merge trees into Huffman Tree





PriorityQueue<HuffmanNode>

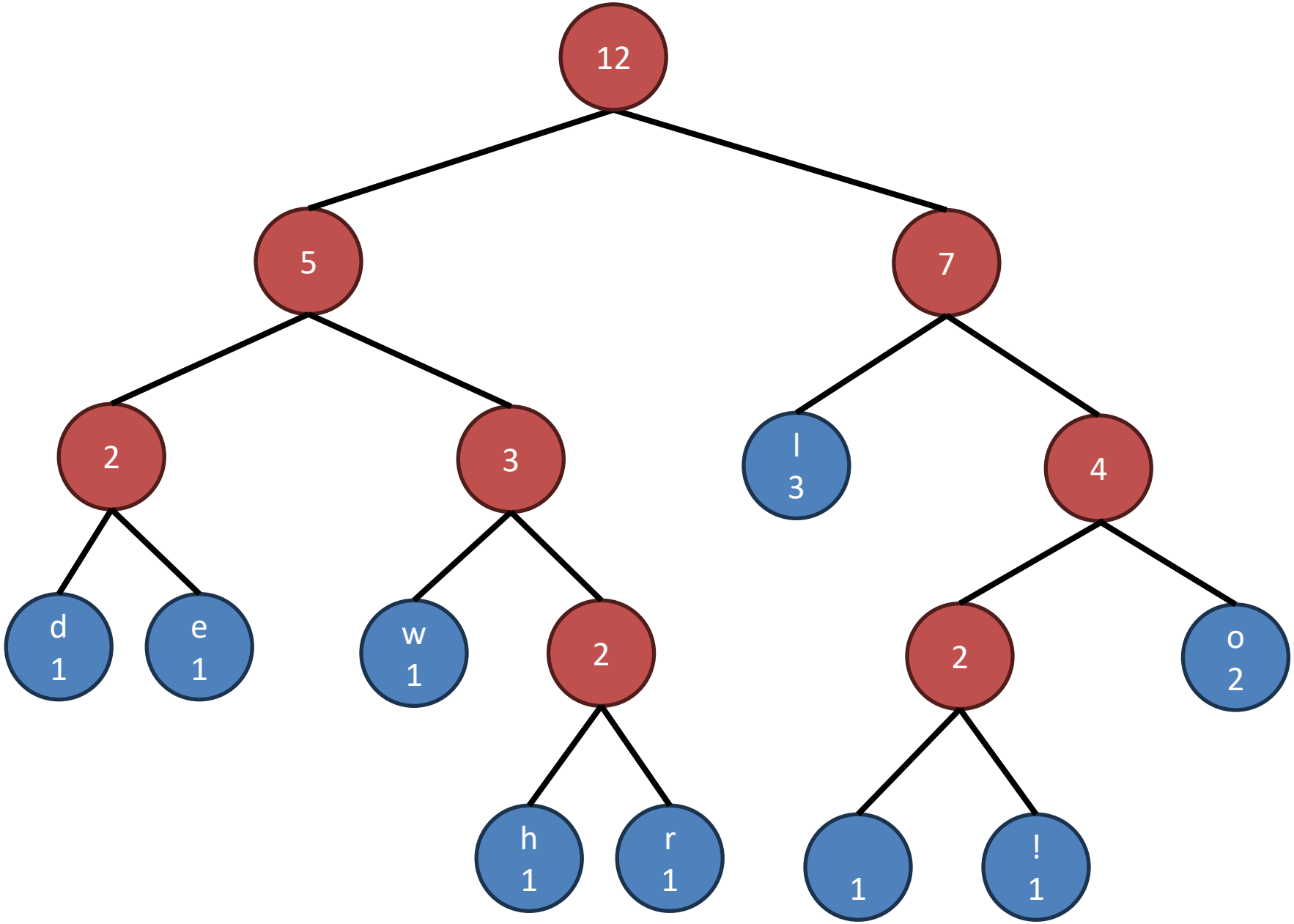
Huffman Coding



12 is still added to the queue, but when there is only one node left, we know there is no more merging to be done!

PriorityQueue<HuffmanNode>

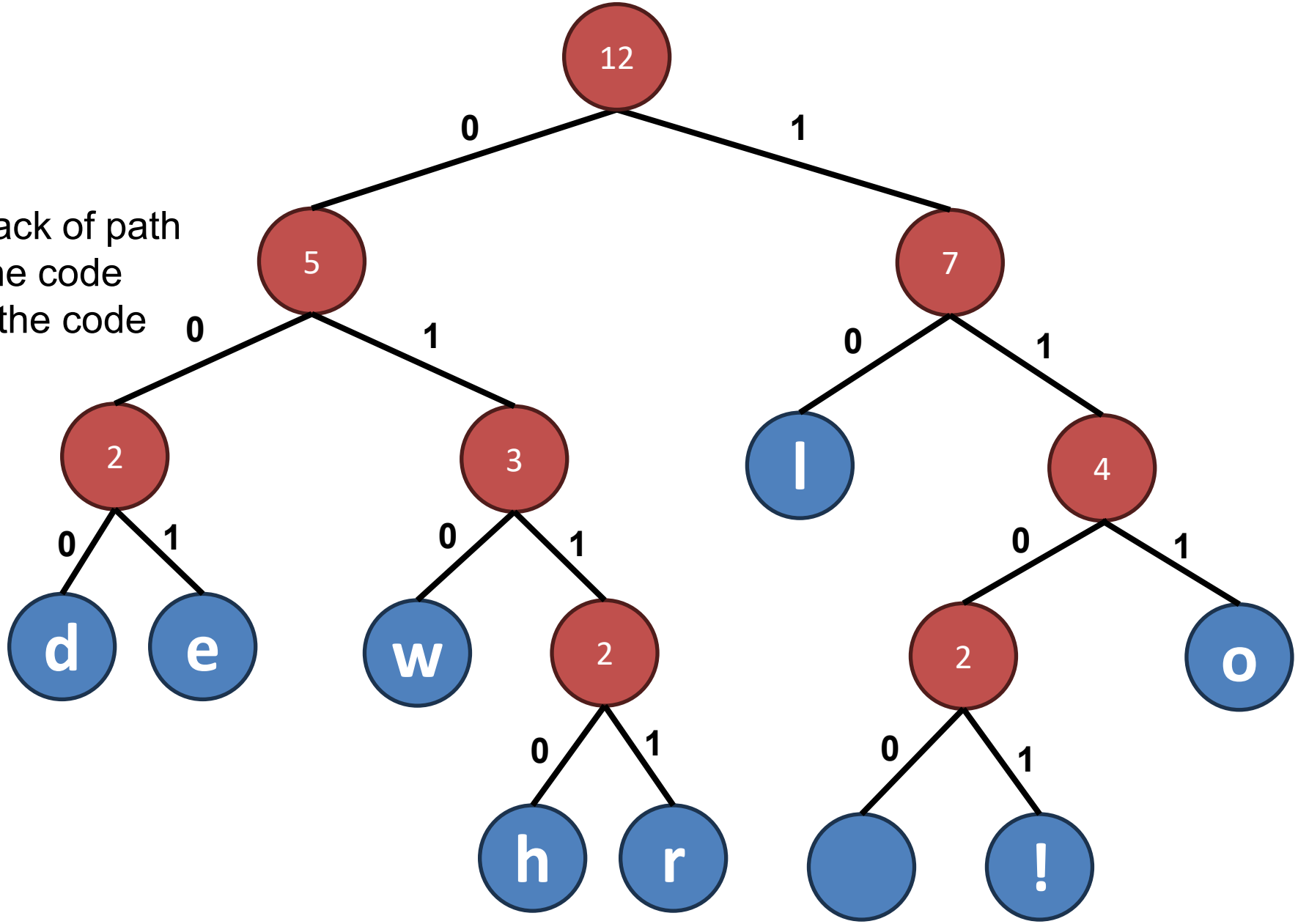
Huffman Coding



Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

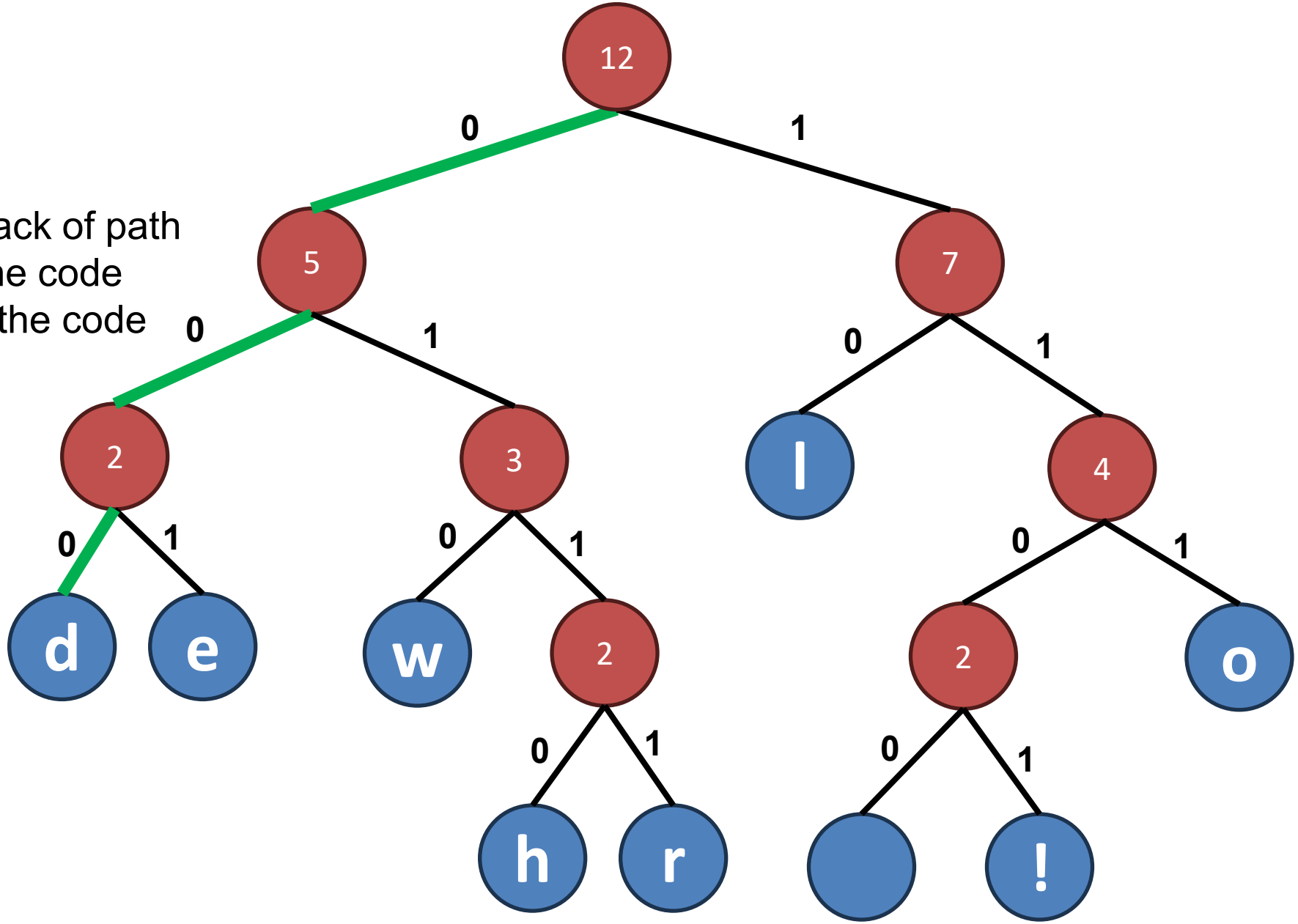


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000

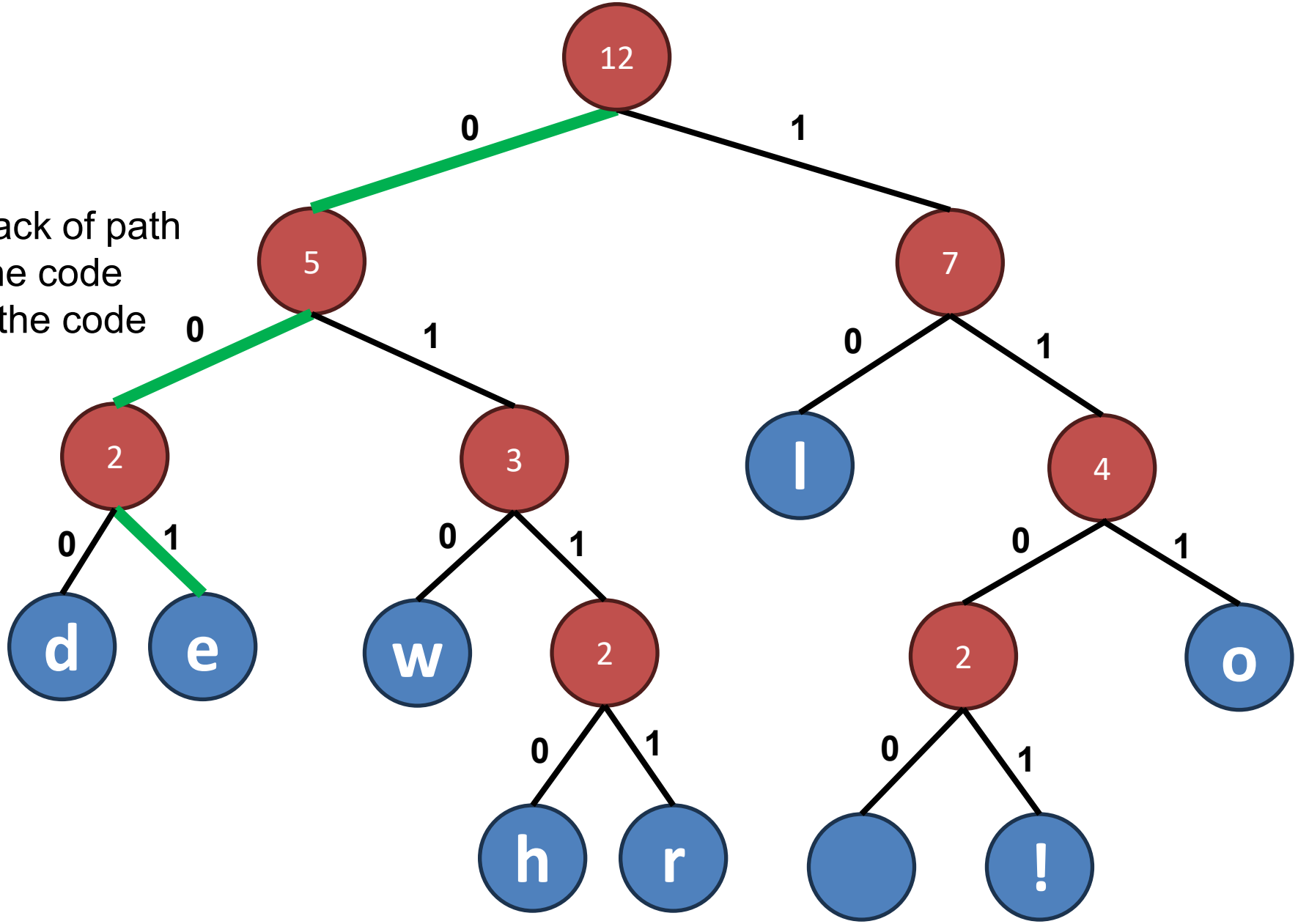


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000
e	001

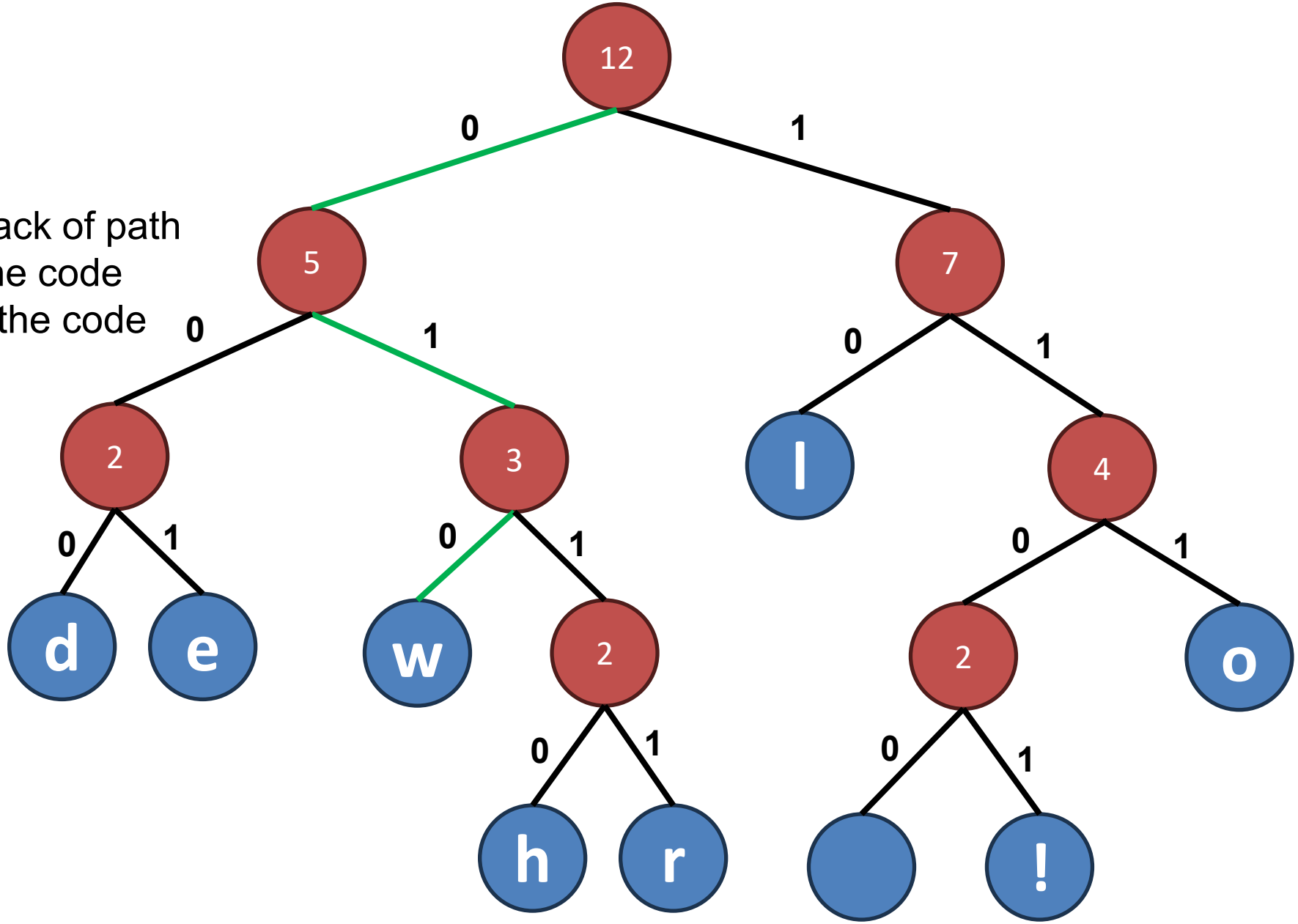


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000
e	001
w	010

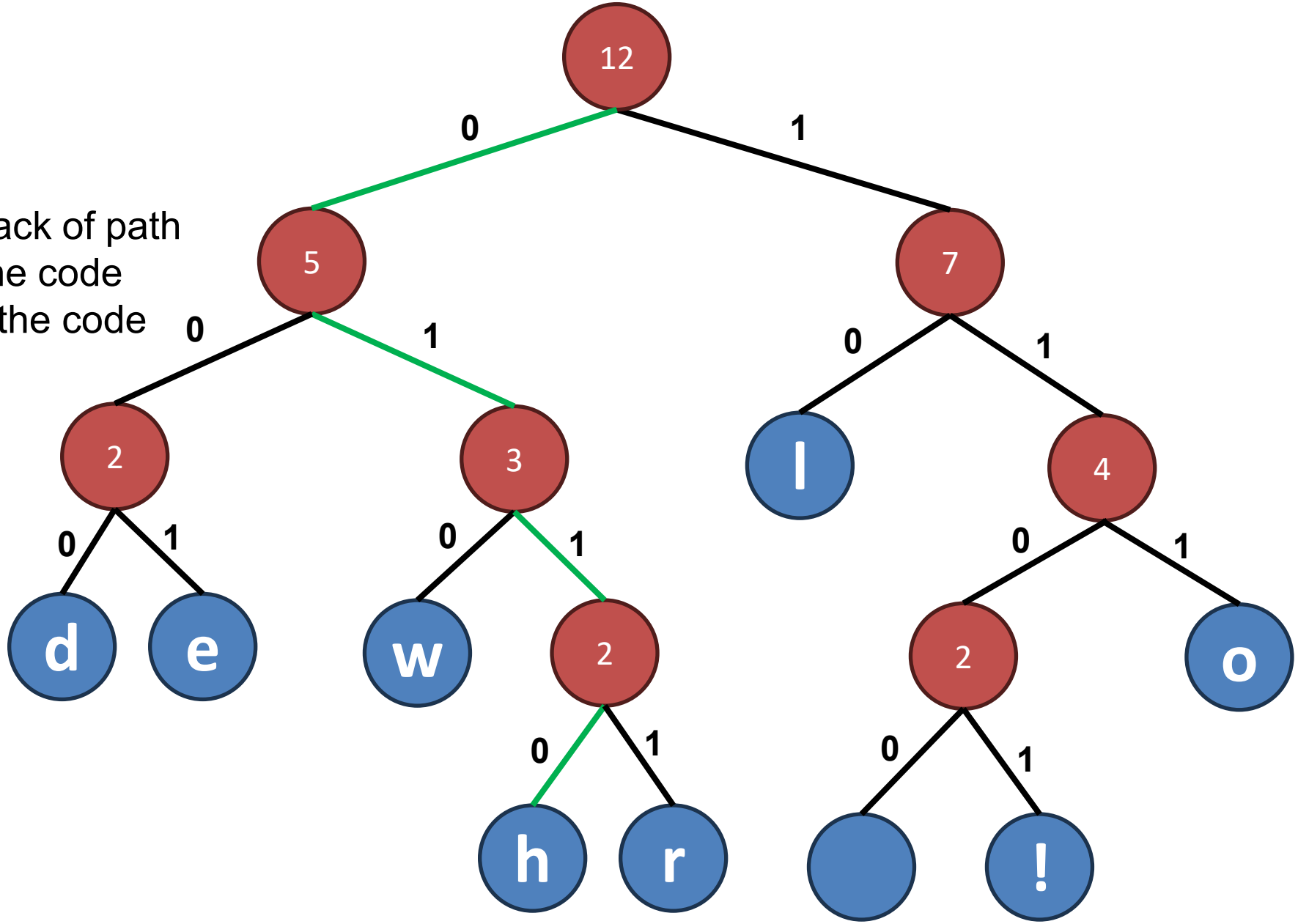


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000
e	001
w	010
h	0110

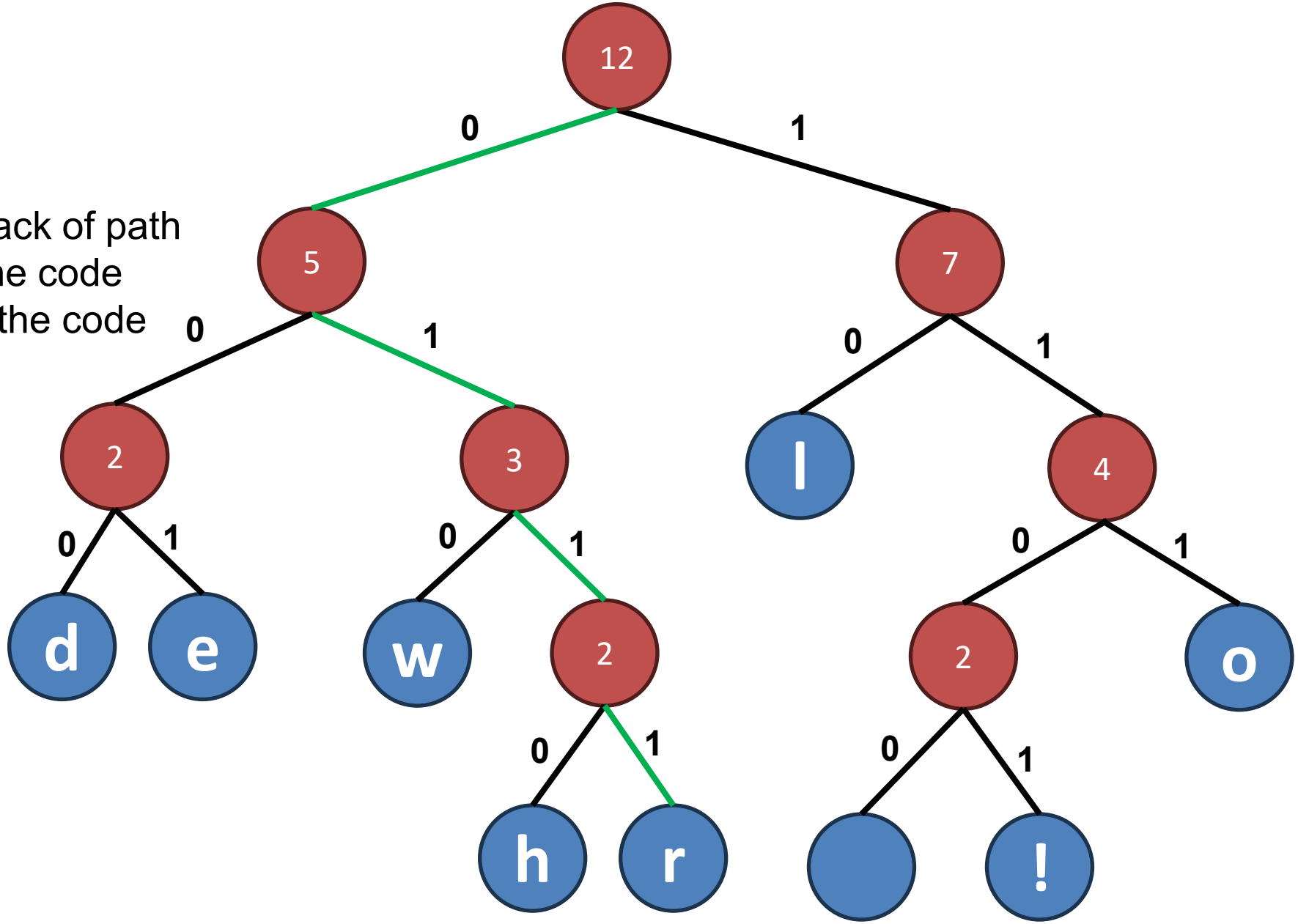


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000
e	001
w	010
h	0110
r	0111

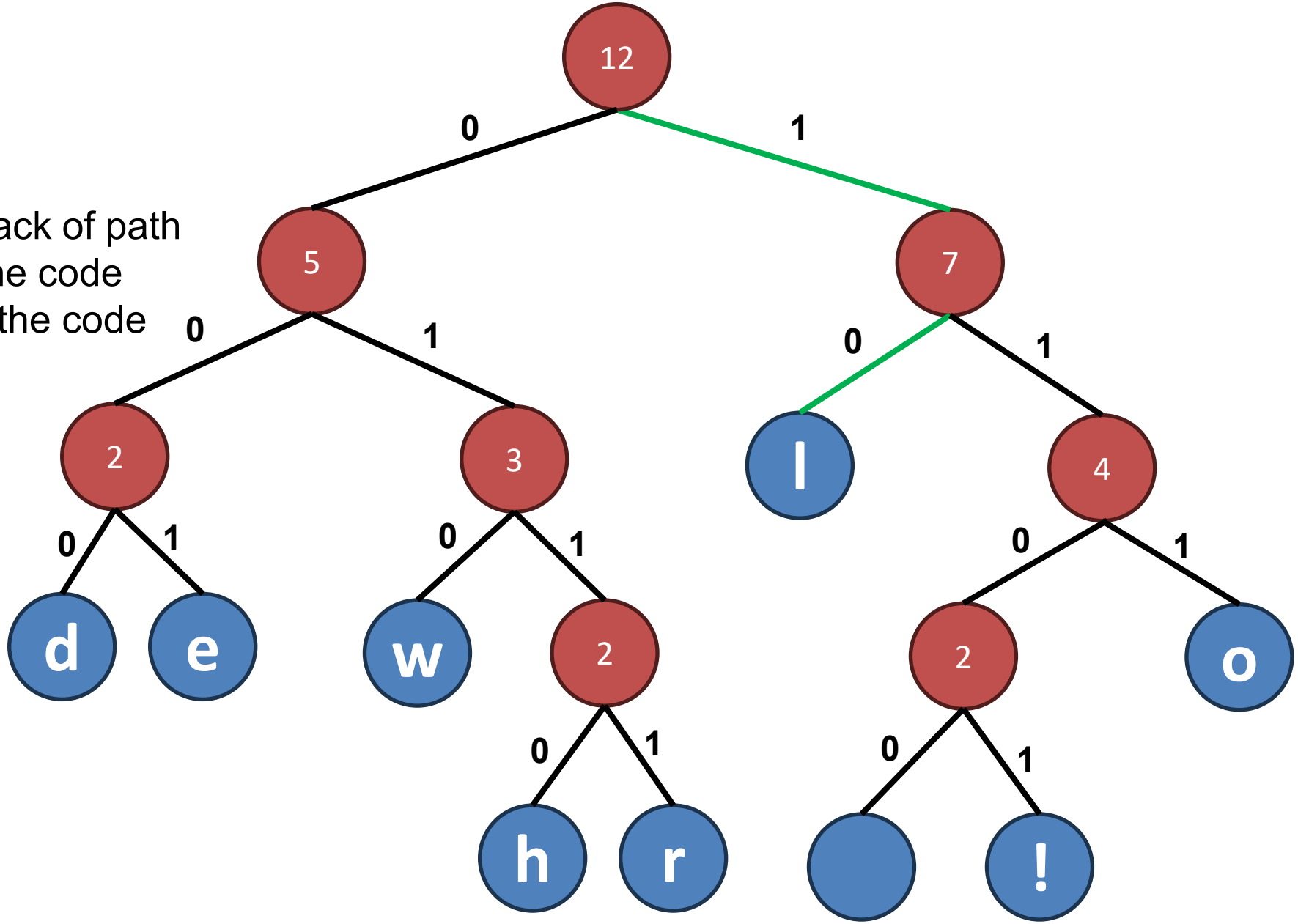


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10

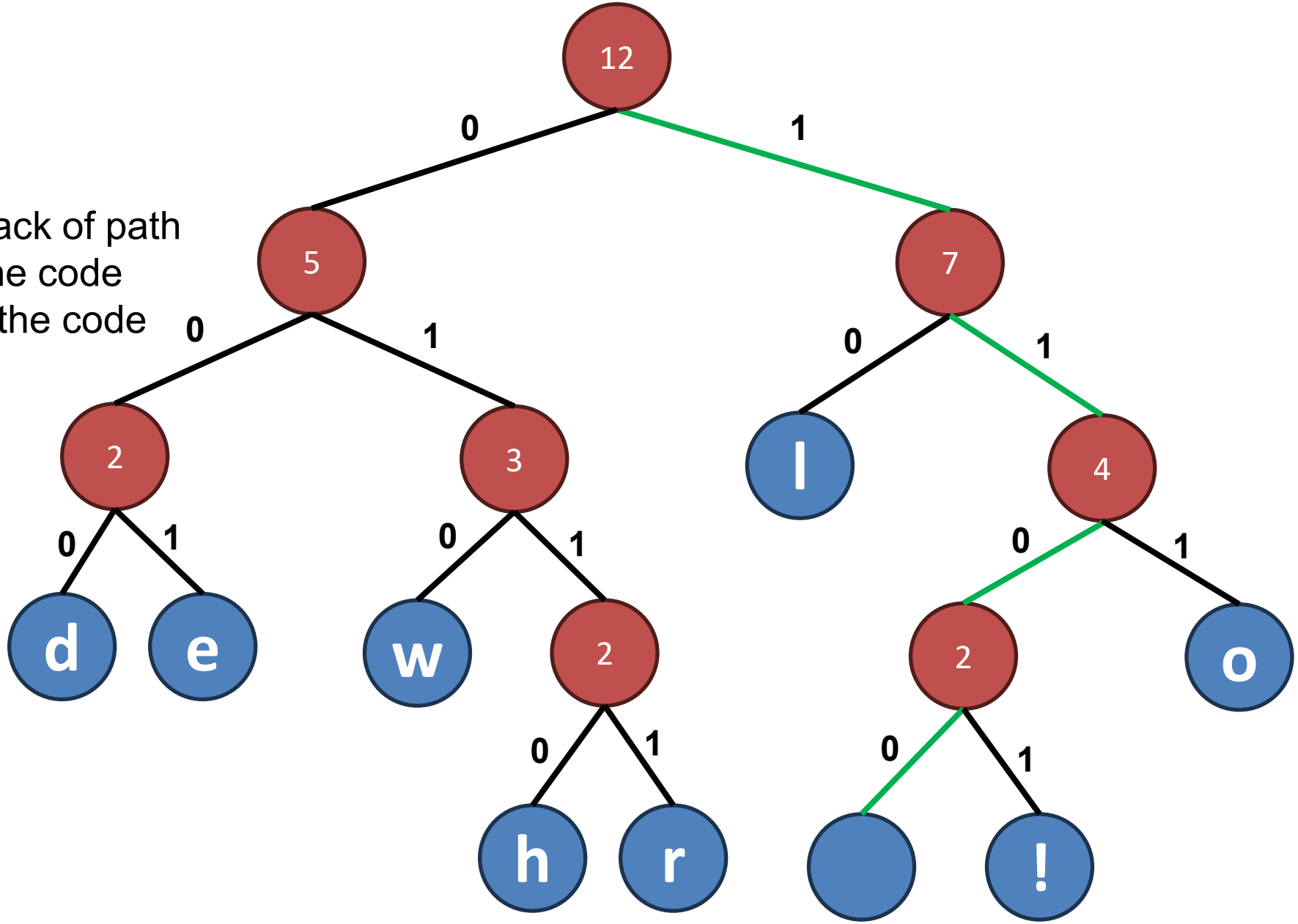


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100

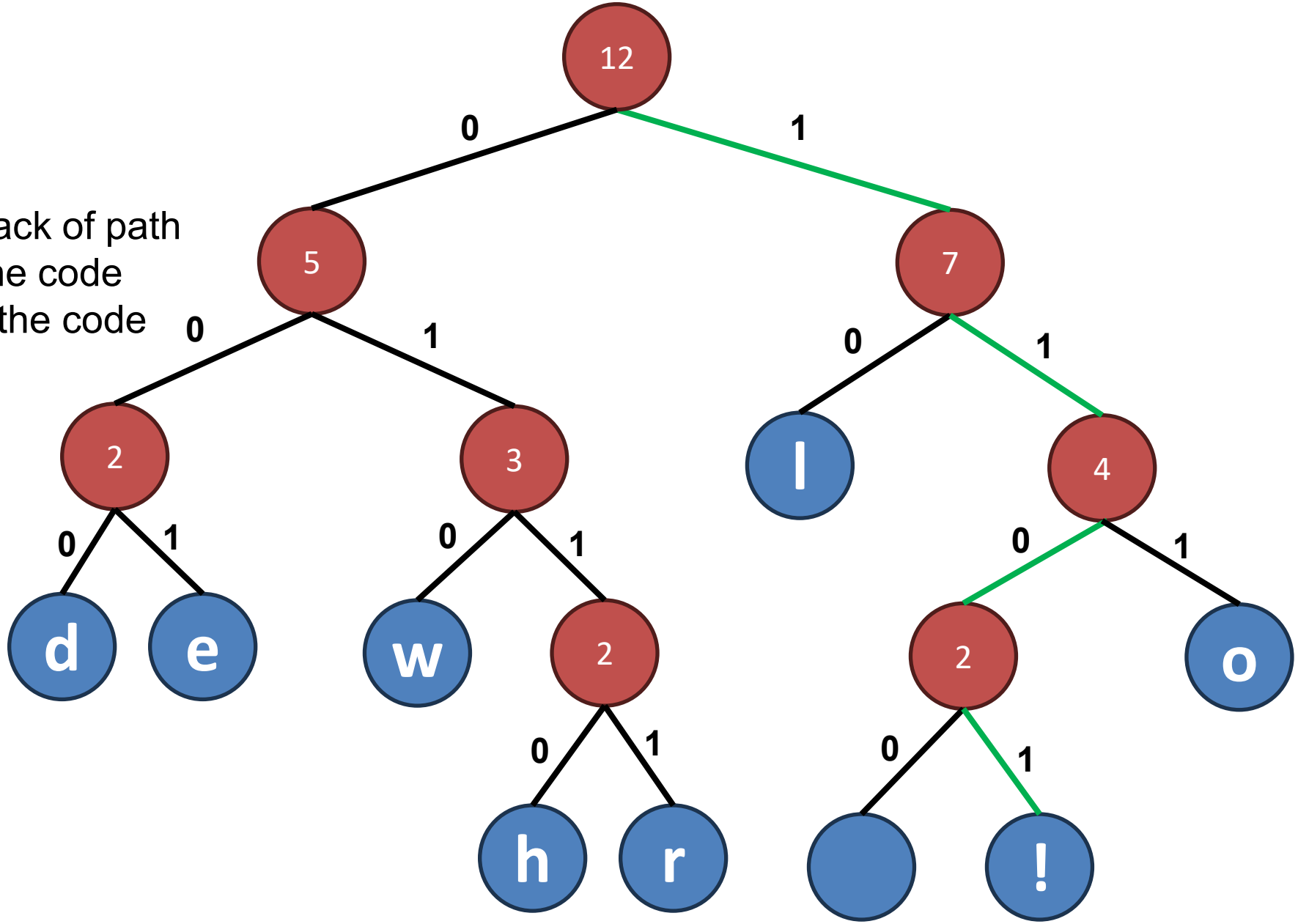


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101

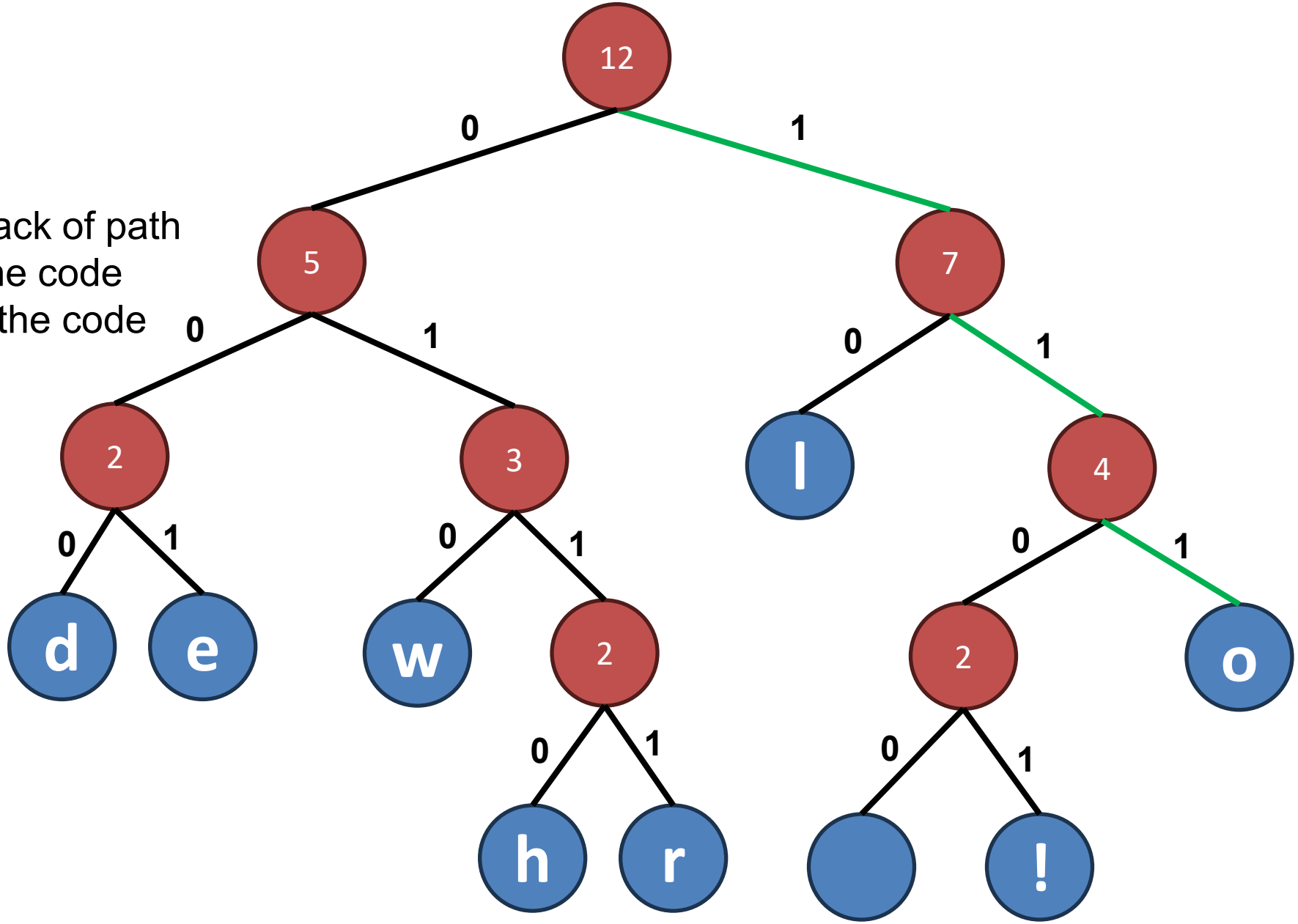


Huffman Coding

Final Step: Generate codes

Visit all leaf nodes and keep track of path
- If we ever go left, add 0 to the code
- If we ever go right, add 1 to the code

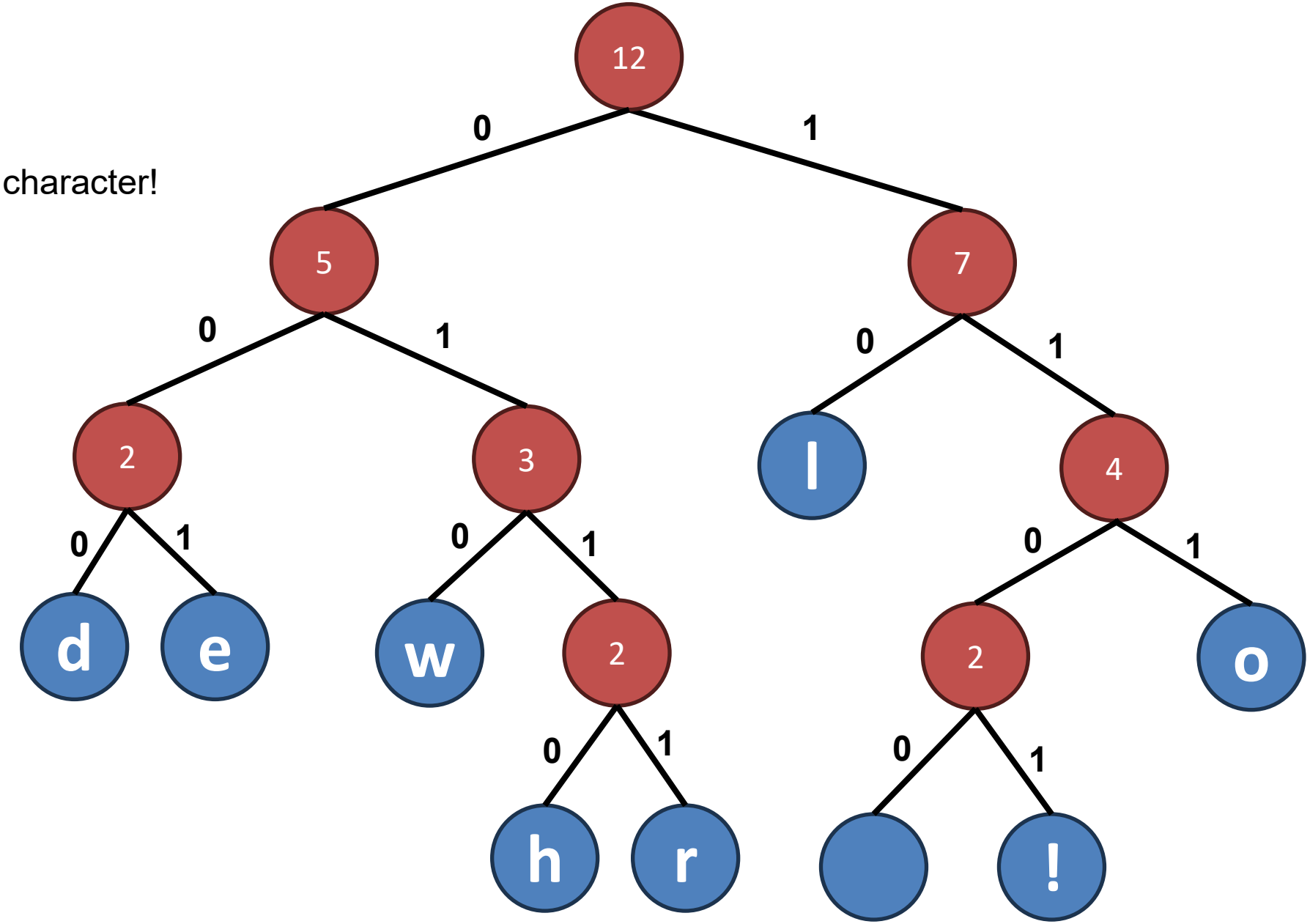
Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111



Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111



Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111

Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111

String: “hello world!”

Encoding w/ Huffman codes:

0110 001 10 10 111 1100 010 111 0111 10 000 1101

Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111

String: “hello world!”

Encoding w/ Huffman codes:

0110 001 10 10 111 1100 010 111 0111 10 000 1101

Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111

String: “hello world!”

Encoding w/ Huffman codes:

0110 001 10 10 111 1100 010 111 0111 10 000 1101

Let’s code !!

Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111

String: “hello world!”

Encoding w/ Huffman codes:

0110 001 10 10 111 1100 010 111 0111 10 000 1101

Message size: 37 bits

Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	001
w	010
h	0110
r	0111
l	10
(space character)	1100
!	1101
o	111

String: “hello world!”

Encoding w/ Huffman codes:

0110 001 10 10 111 1100 010 111 0111 10 000 1101

Message size: 37 bits

Encoding w/ UTF:

01101000 01100101 01101100 01101100 01101111 00100000
01110111 01101111 01110010 01101100 01100100

Message size: 88 bits

Huffman Coding

We now have encodings for each character!

Character	Code
d	000
e	
h	
r	
l	
(space character)	1100
!	1101
o	111

Huffman Coding is a **compression** algorithm

…rld!”

Huffman codes:

001 10 10 111 1100 010 111 0111 10 000 1101

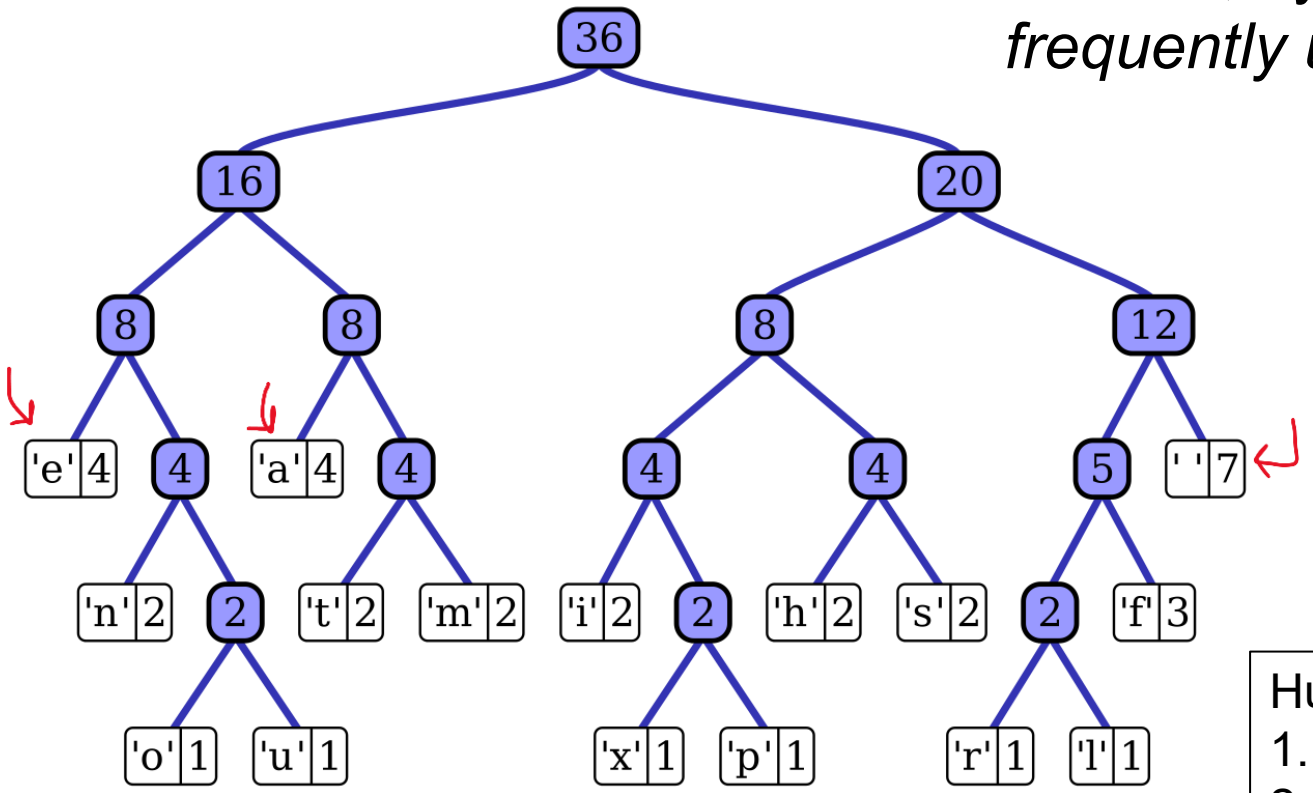
Message size: 37 bits

Encoding w/ UTF:

01101000 01100101 01101100 01101100 01101111 00100000
01110111 01101111 01110010 01101100 01100100

Message size: 88 bits

Huffman Coding is a way to encode a string using a binary tree



It also acts as a lossless compression algorithm for data, by creating smaller encodings for *frequently used characters*

When we compress files with formats such as `.zip`, `.7z`, `.rar` , **Huffman Coding** is used to compress the data!

It is also used for image, video, audio file compression, and even fax machines !

Huffman Coding Algorithm
1. Put nodes in PQ **O(n)**
2. Extract two smallest nodes and merge (repeated n times) } **O(nlogn)**
3. Iterate through to all leaf nodes and get encoding **O(n)**

Total Running time: O(nlogn) where n = # of characters

Huffman Coding Visualization

https://cmps-people.ok.ubc.ca/ylucet/DS/Huffman.html