Huffman Coding

Thomas Przybylinski

Character Encodings

- The usual encodings in computer are usually either ASCII or UNICODE.
- ASCII is a fixed-width encoding of 7 bits, though its usually stored in 8.
- For most intents and purposes, UNICODE has a fixed-width encoding of 16 bits.

Variable Length Encodings

- A different way to encode characters is to use a variable-length encoding.
- The characters could be encoded in a mixture of 1 bit, 2 bits, 3 bits, etc
- However, to avoid confusion, variable-length encodings need to be prefix-free
    - This means no encoding for one character can be the prefix of the other.
    - This is to avoid ambiguity

Prefix-Free

- Let's say we have this encoding:
  - E=01, N=100, O=11, S=1 and Y=10
- How would you decode:
  - 10011

It could be either of:
100 11 “NO” or
10 01 1 “YES”

Variable Length Encodings

- In many cases, certain characters are used more often than others.
- In English usage, e occurs more frequently than other letters. Lower case occurs more frequently than upper case. ASCII 7 (the bell) is rarely if ever used.
- So perhaps we could save some space if we encode more-frequent characters in the smaller encodings.

	Fixed Width	Variable Width
A	0 0	0
B	0 1	1 0
C	1 0	1 1 0
D	1 1	1 1 1

Compared to the fixed-width encoding:
A uses 1 less bit
C and D use 1 more bits each.

So for there to be space savings:
A must be more frequent than C and D combined.

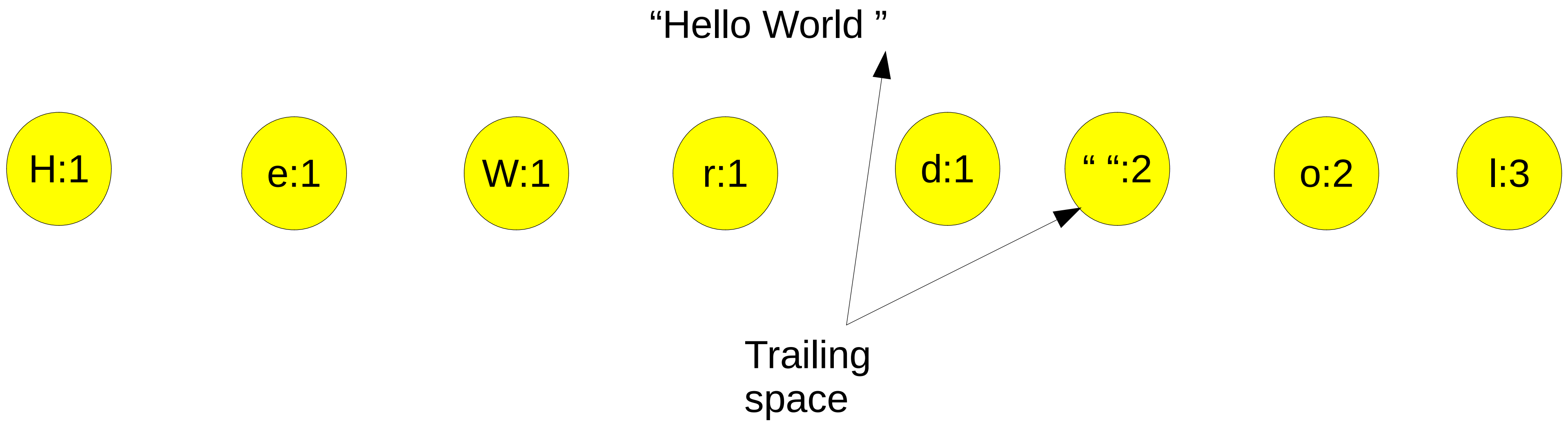
Assume in a given document, there is a 90% chance a given character is A, 10% chance of B, 5% chance of C and 5% chance of D.

Let's see what the average encoding length would be:
Fixed-Width length = 2
Variable-Width length = .9*1 +.1*2 + .05*3 + .05*3
= .9 + .2 + .15 + .15
= 1.4, which is about 30% less than the fixed-width

Huffman Coding

- Huffman Coding is a greedy algorithm to try and find a good variable-length encoding given character frequencies.

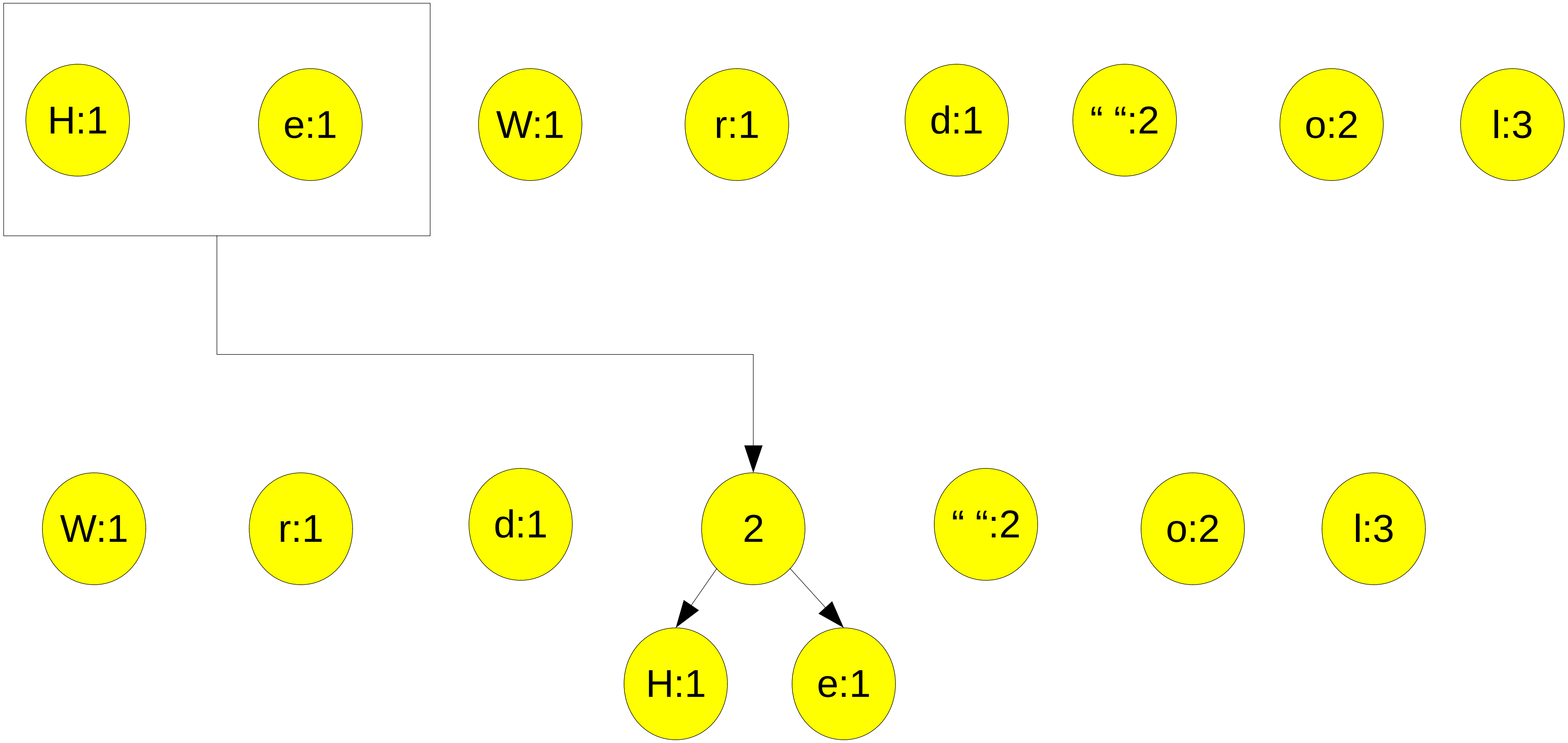
- In the algorithm, we are going to create larger binary trees from smaller trees.
- Initially, our smaller trees are single nodes that correspond to characters and have a frequency stored in them


“Hello World ”
H:1
e:1
W:1
r:1
d:1
“ “:2
o:2
l:3
Trailing
space

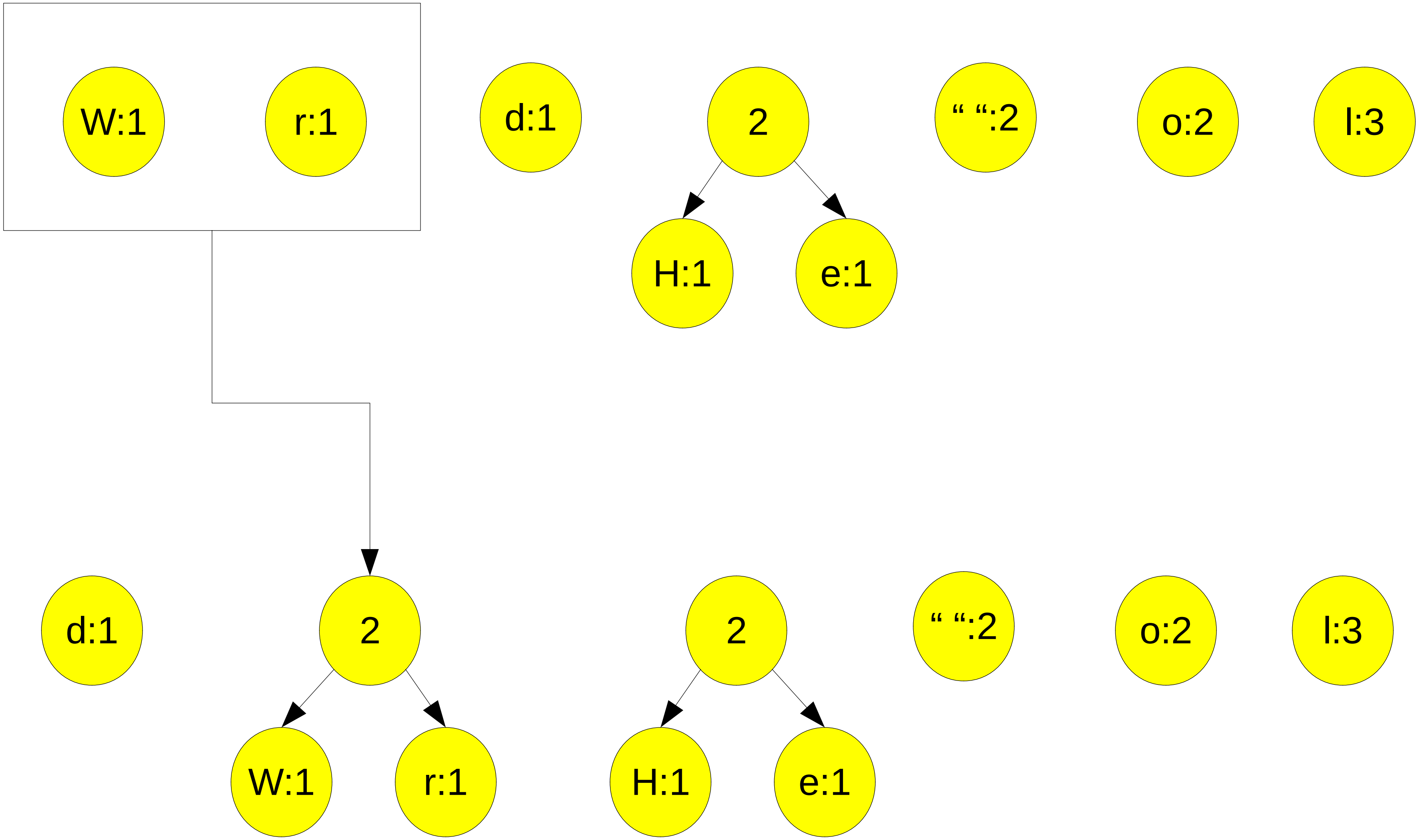
Tree Growing Step

- We take the two trees with roots of smallest frequency (tied broken arbitrarily) and merge them.
- The merge operation takes the two trees, creates a node whose key is the sum of the frequencies of the two roots nodes, and make that node the new root.
- The best way to get the two smallest trees is with a priority queue, using poll() twice, adding the merged tree back into the priority queue.

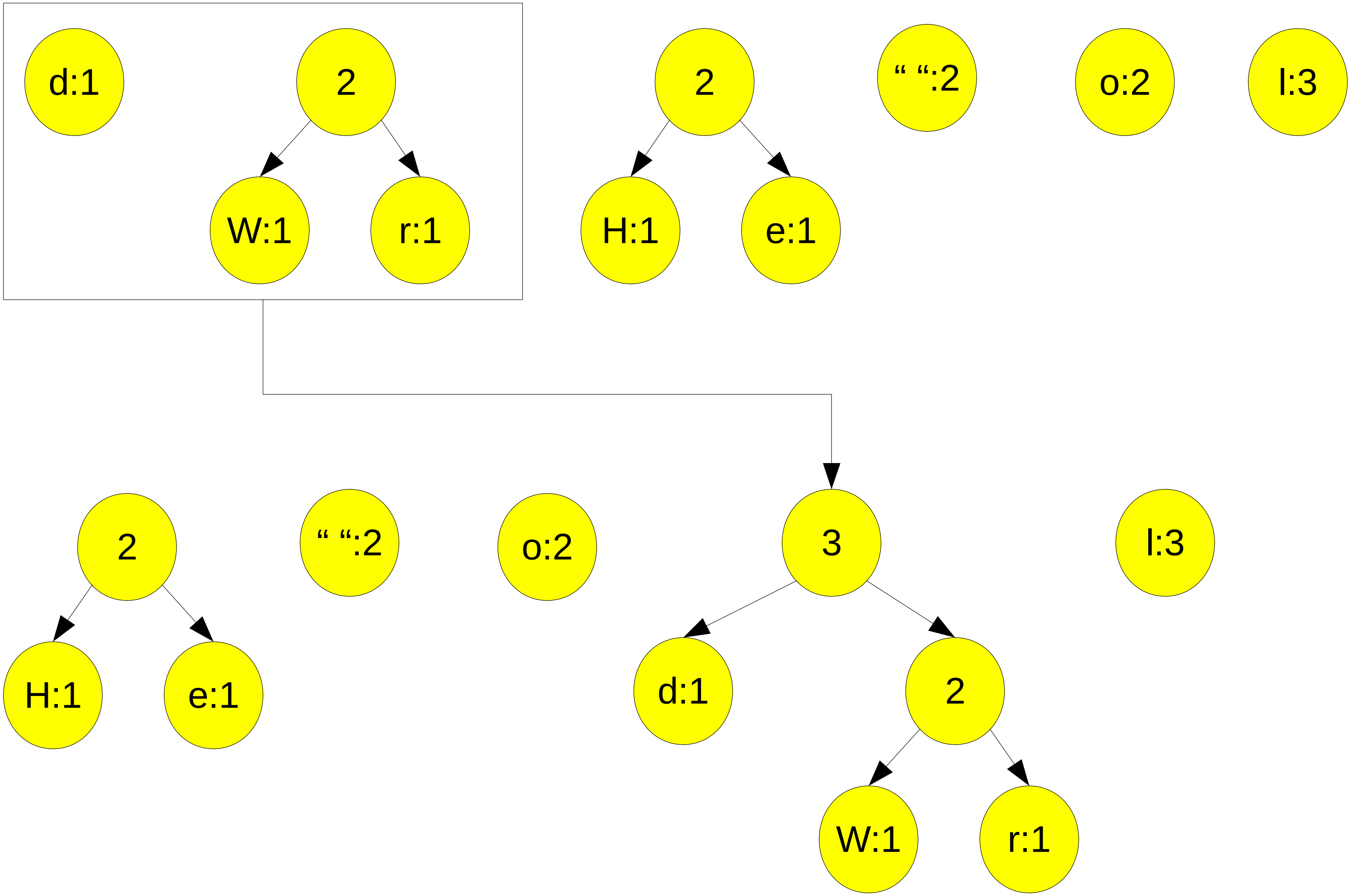
“Hello World ”



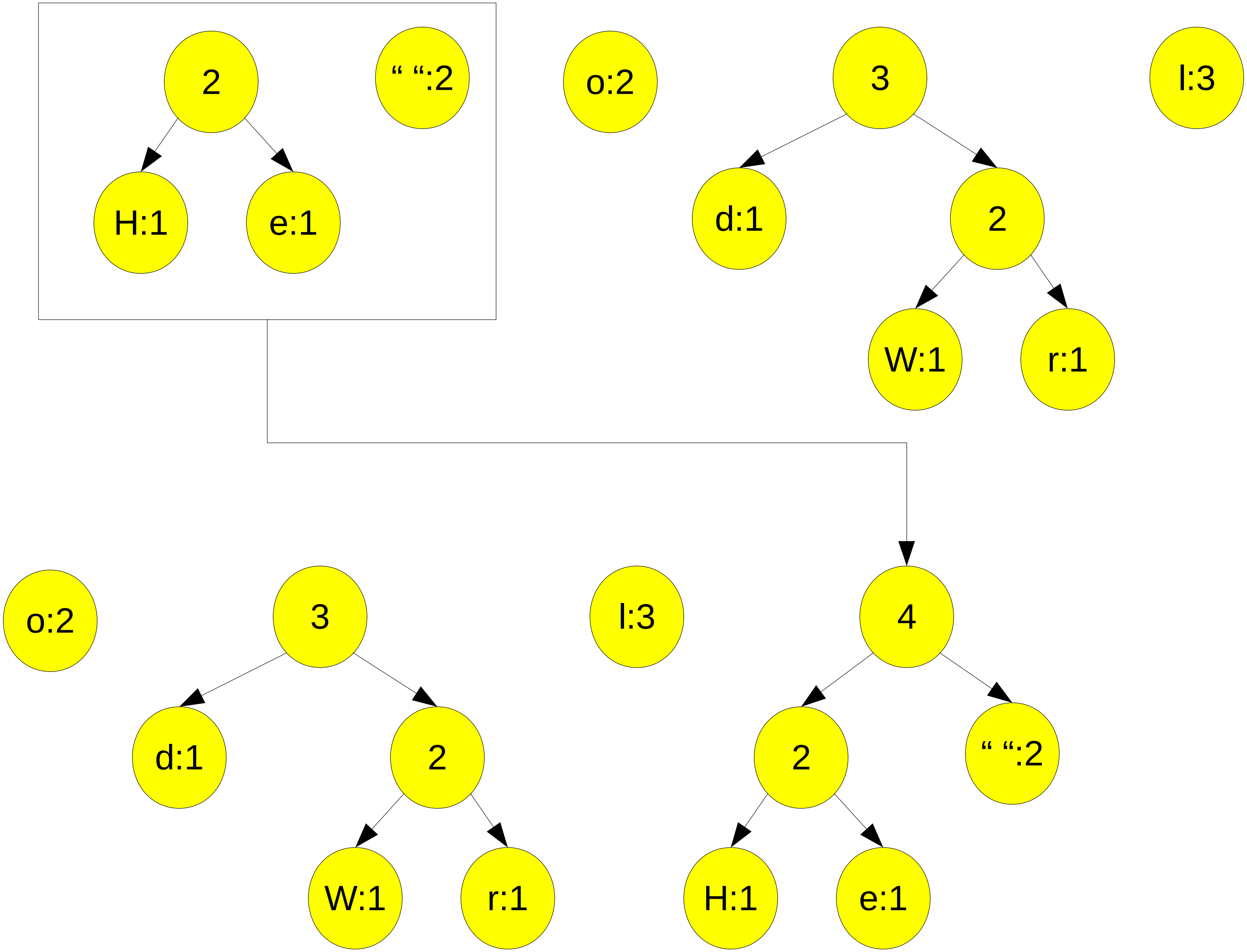
“Hello World ”



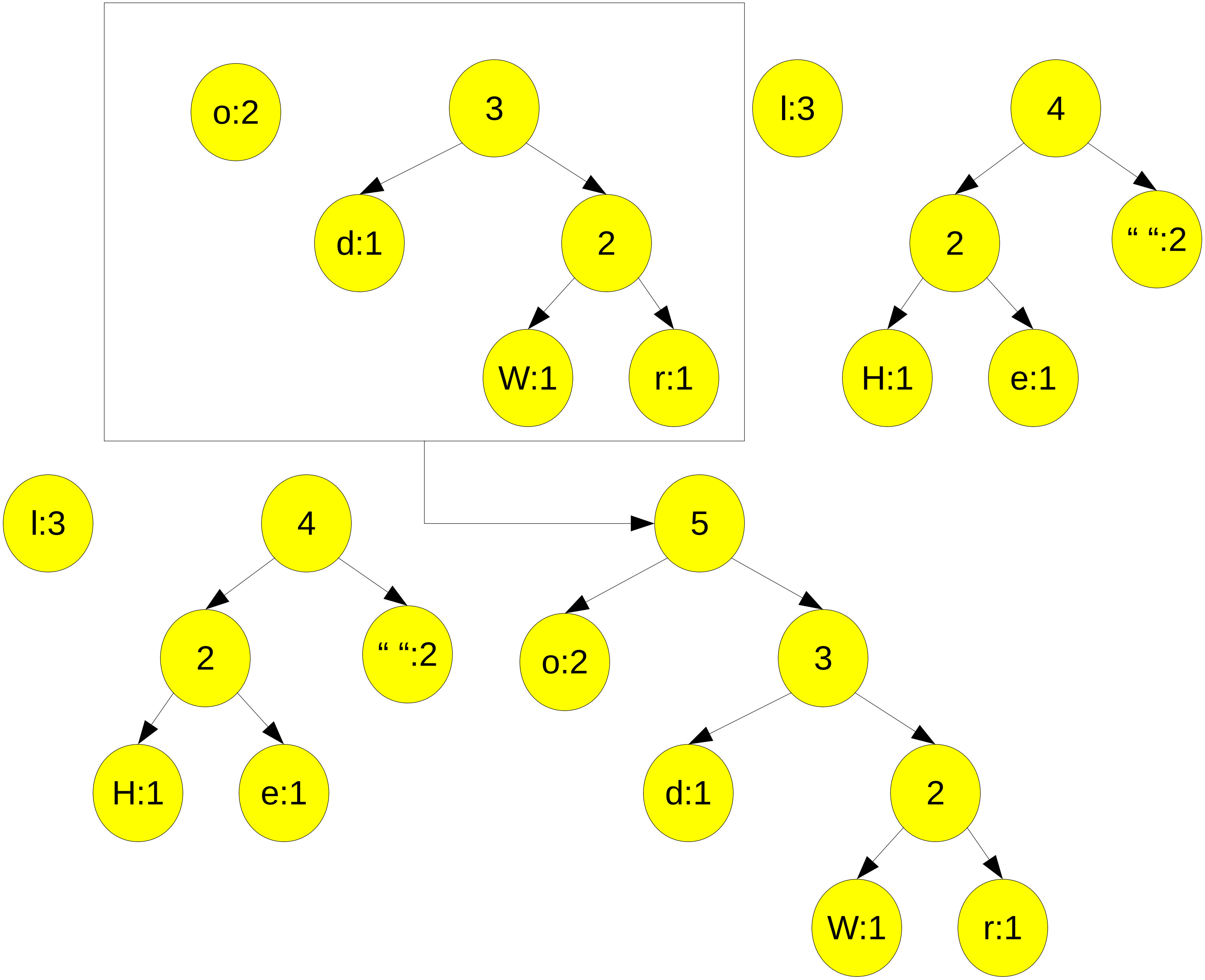
“Hello World ”

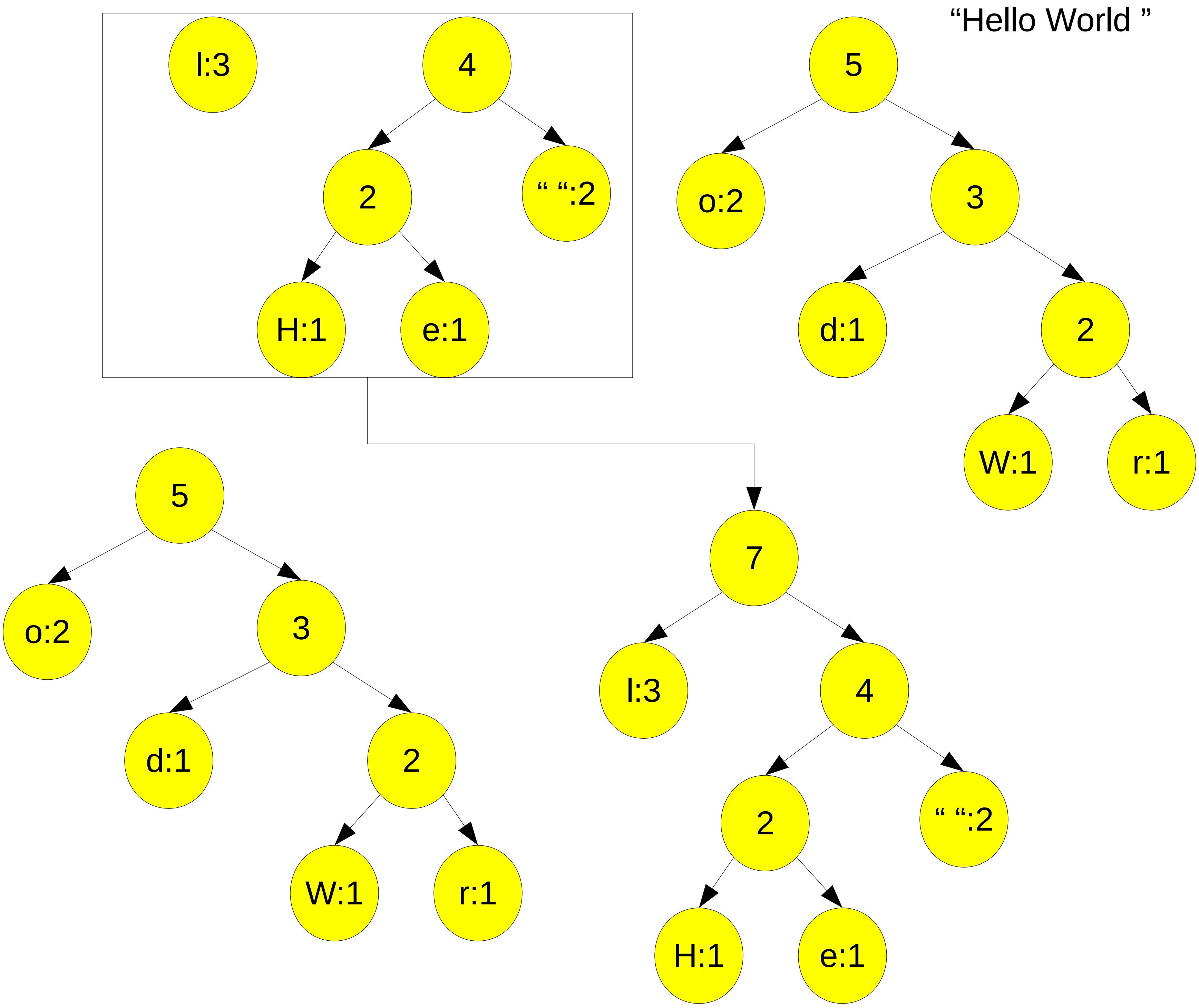


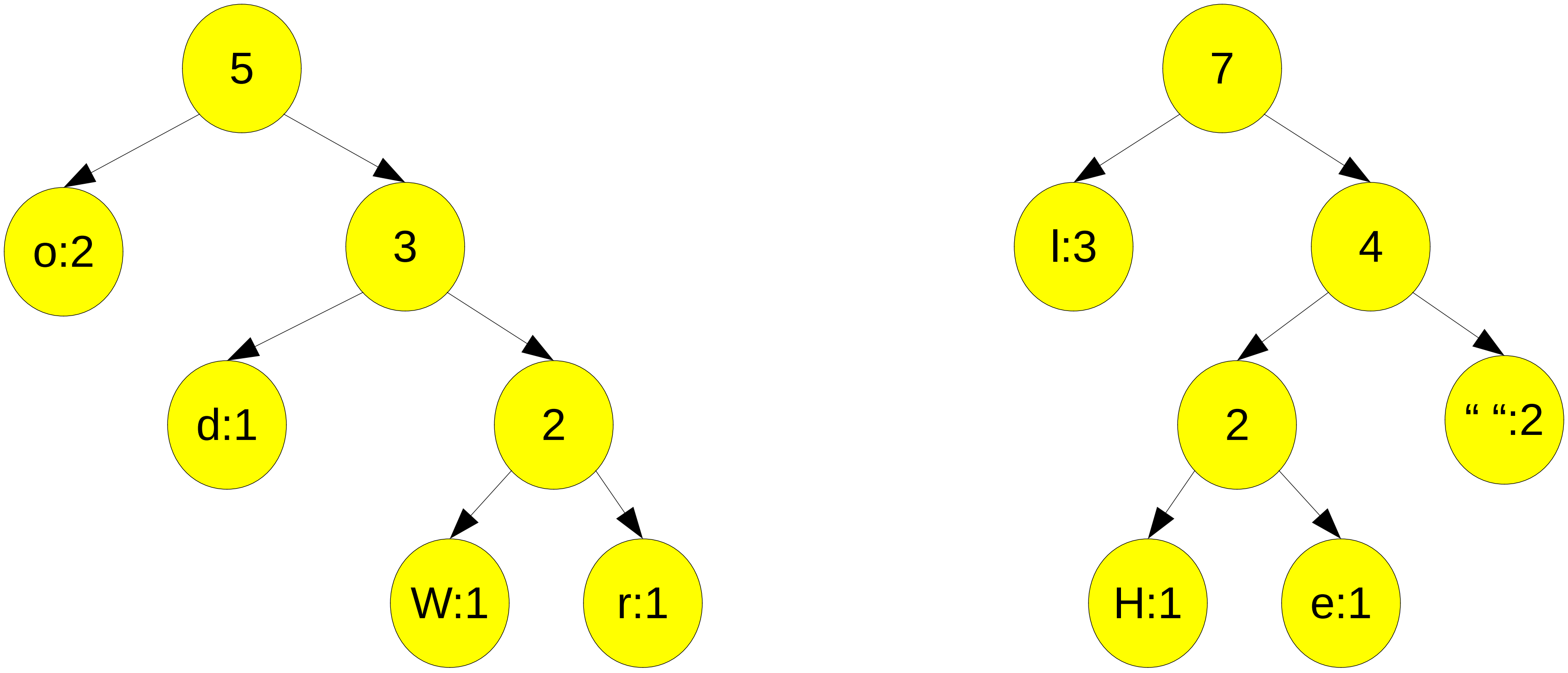
“Hello World ”

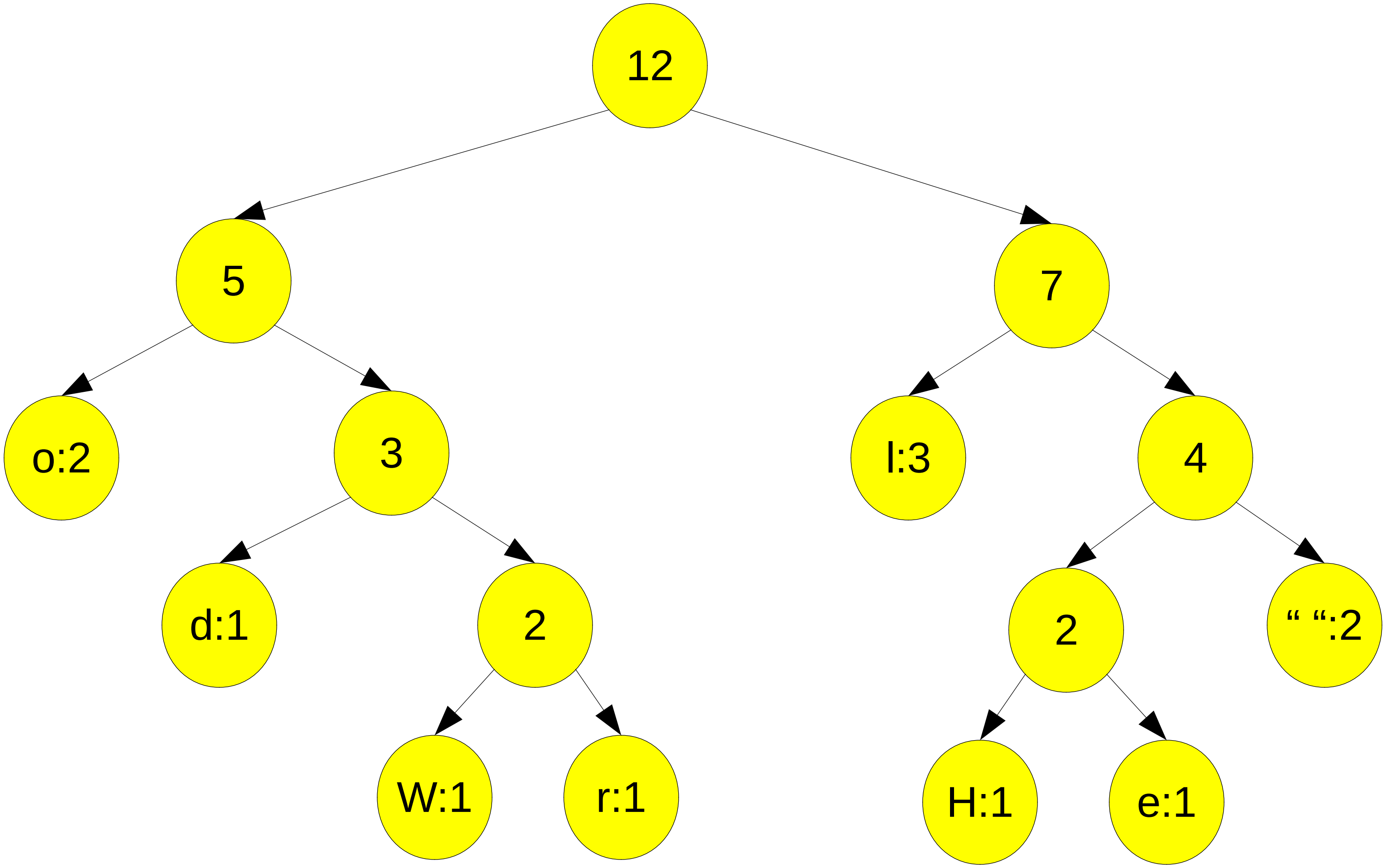


“Hello World ”



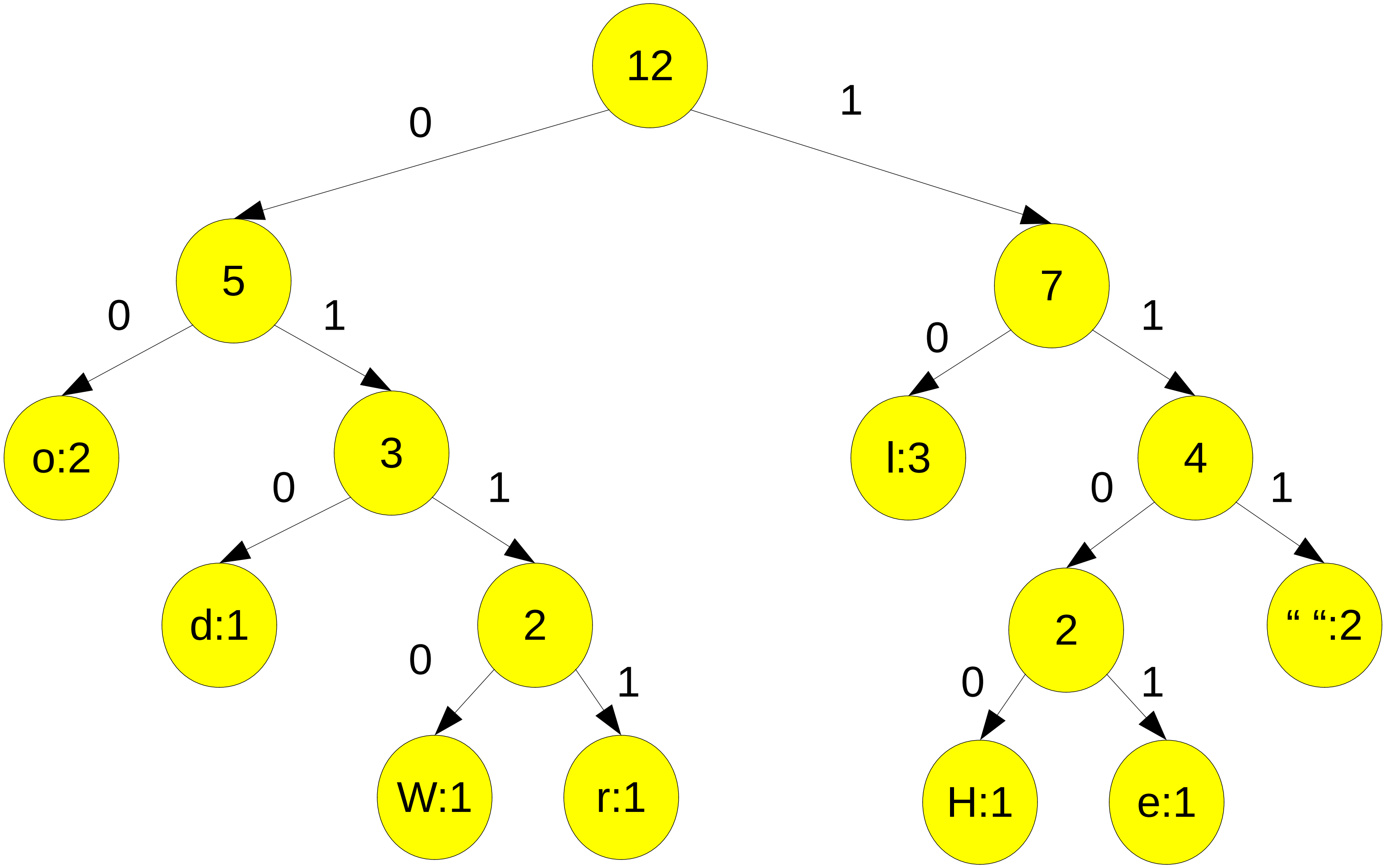

“Hello World ”
l:3
4
2
“ “:2
H:1
e:1
5
o:2
3
d:1
2
W:1
r:1
5
o:2
3
d:1
2
W:1
r:1
7
l:3
4
2
“ “:2
H:1
e:1


5
o:2
3
d:1
2
W:1
r:1
7
l:3
4
2
" ":2
H:1
e:1

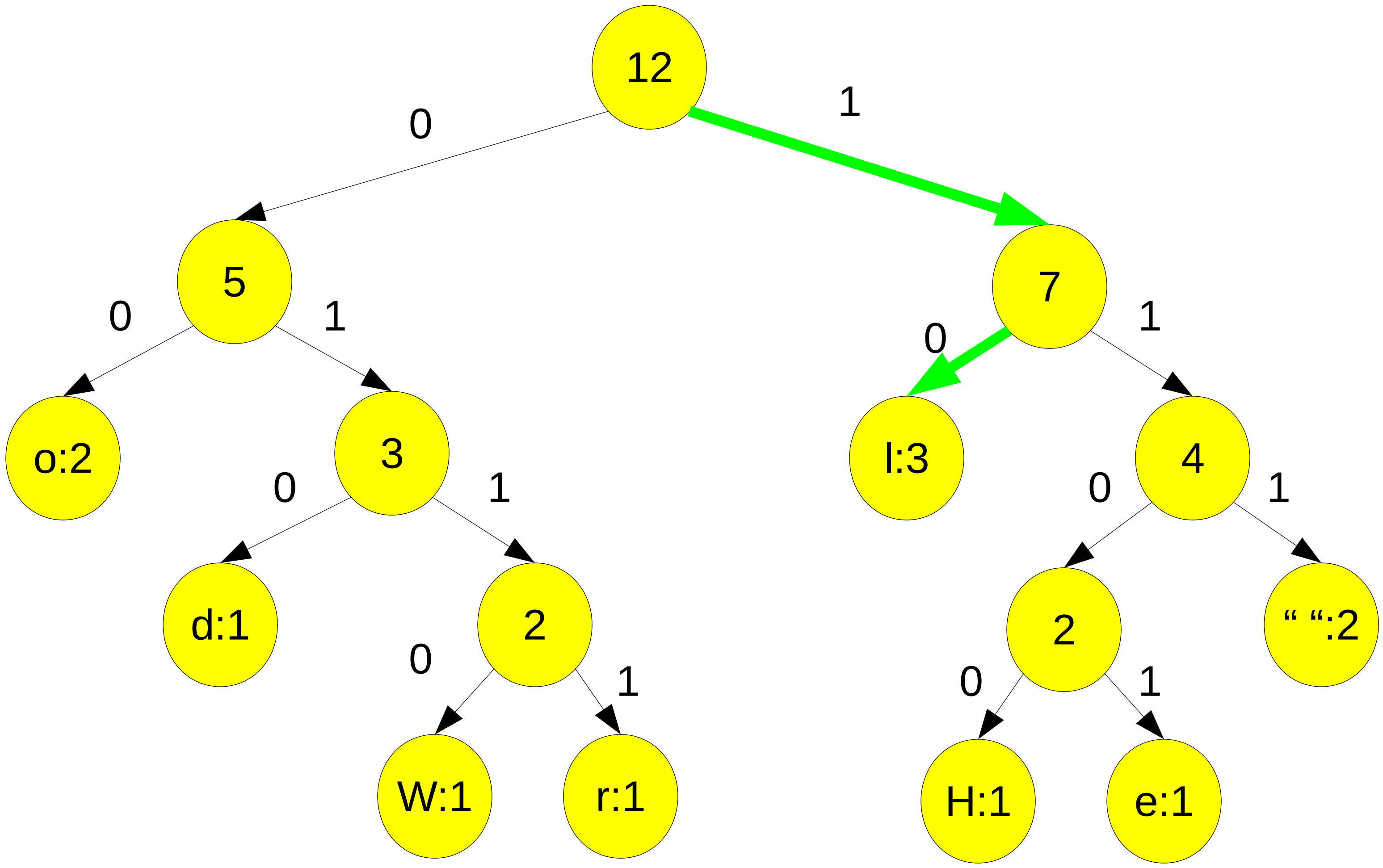

12
5
7
o:2
3
l:3
4
d:1
2
2
“ “:2
W:1
r:1
H:1
e:1

Assigning Bits

- Once we have one tree, we need to make the encoding.
- We do this by encoding the tree traversal from the root to the character node.
    - Every time we go left, we add a 0
    - Every time we go right, we add a 1

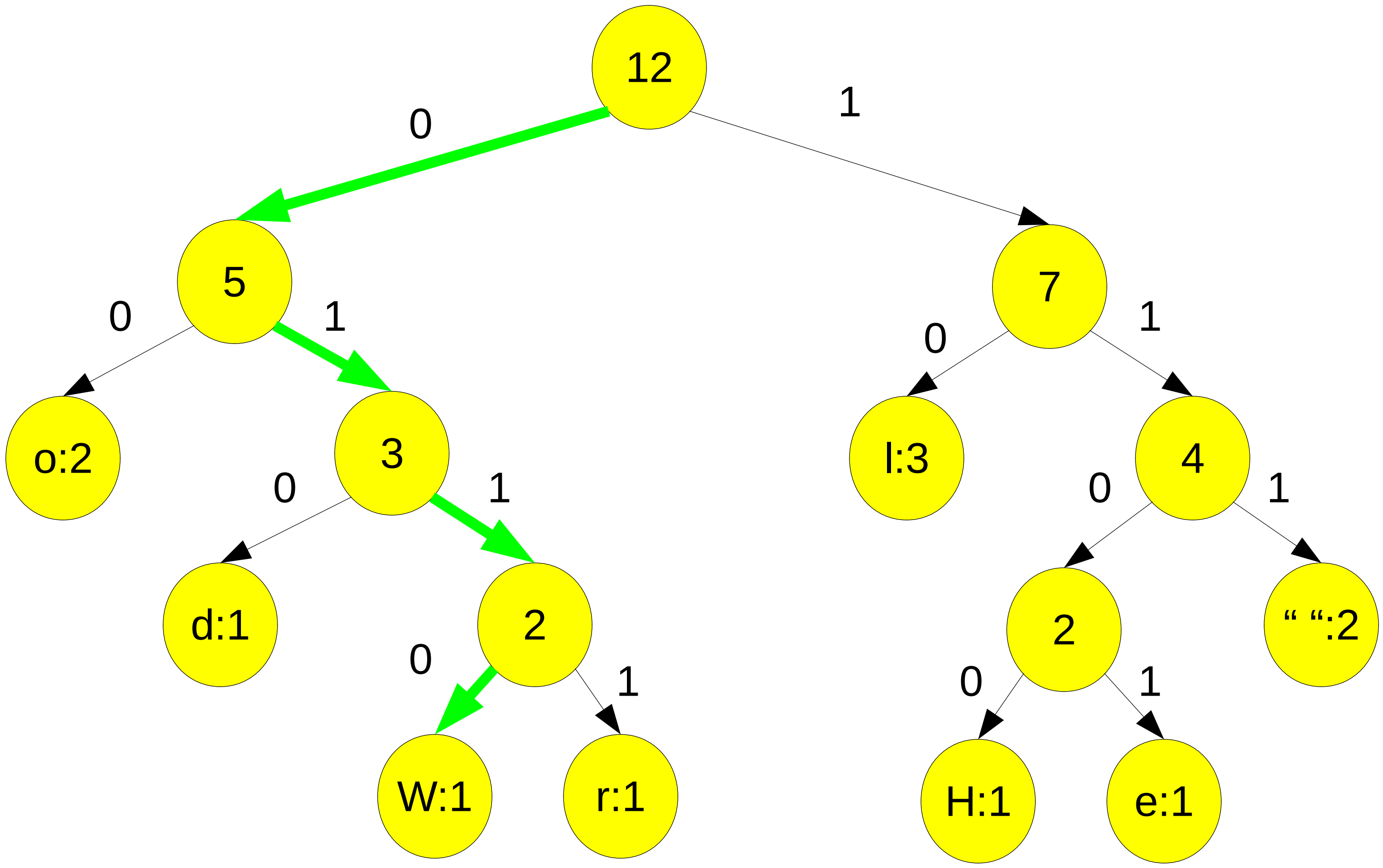

12
0
1
5
7
0
1
0
1
o:2
3
l:3
4
0
1
0
1
d:1
2
2
" ":2
0
1
0
1
W:1
r:1
H:1
e:1

What is the code for l?



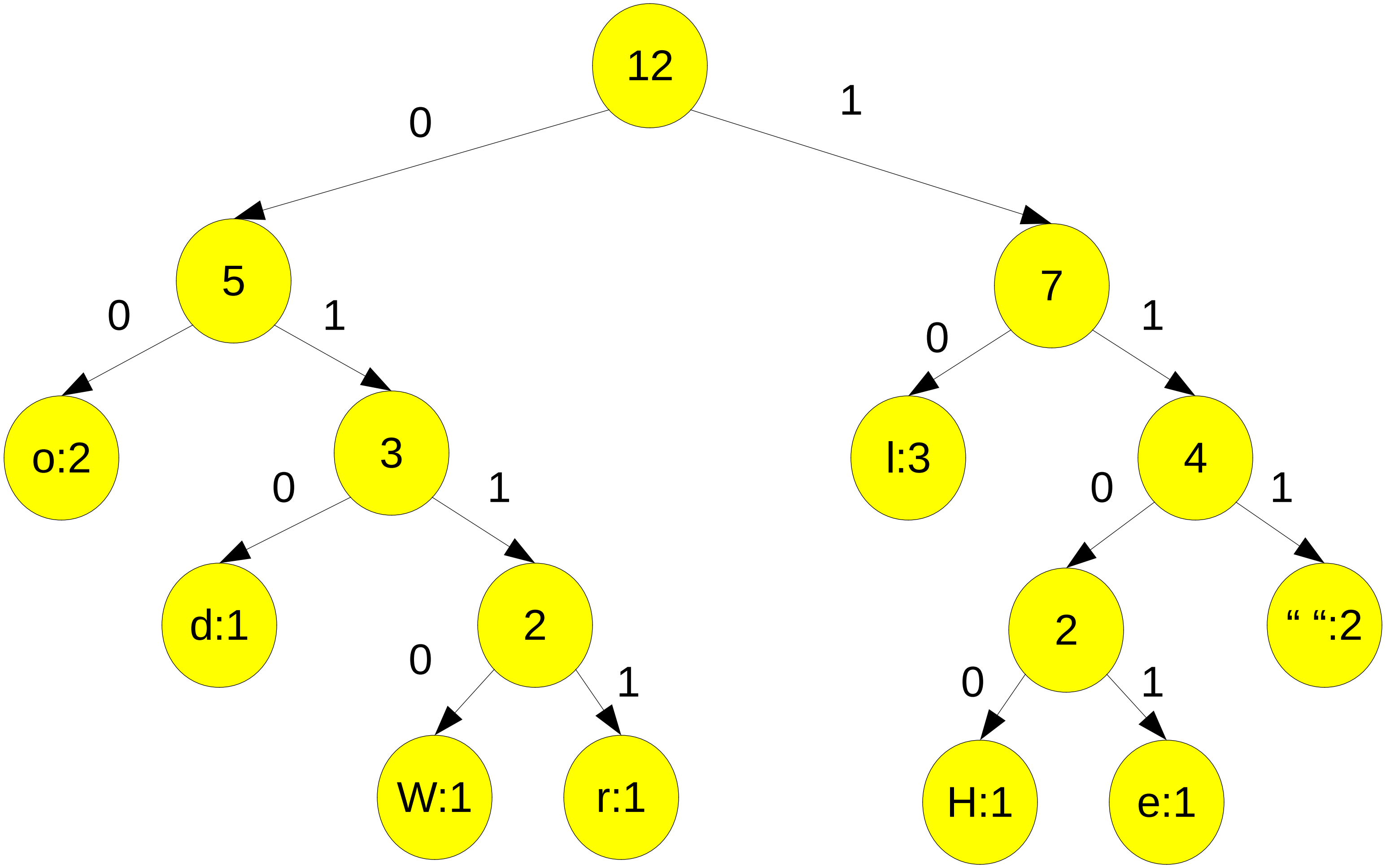
10

What is the code for W?



0110

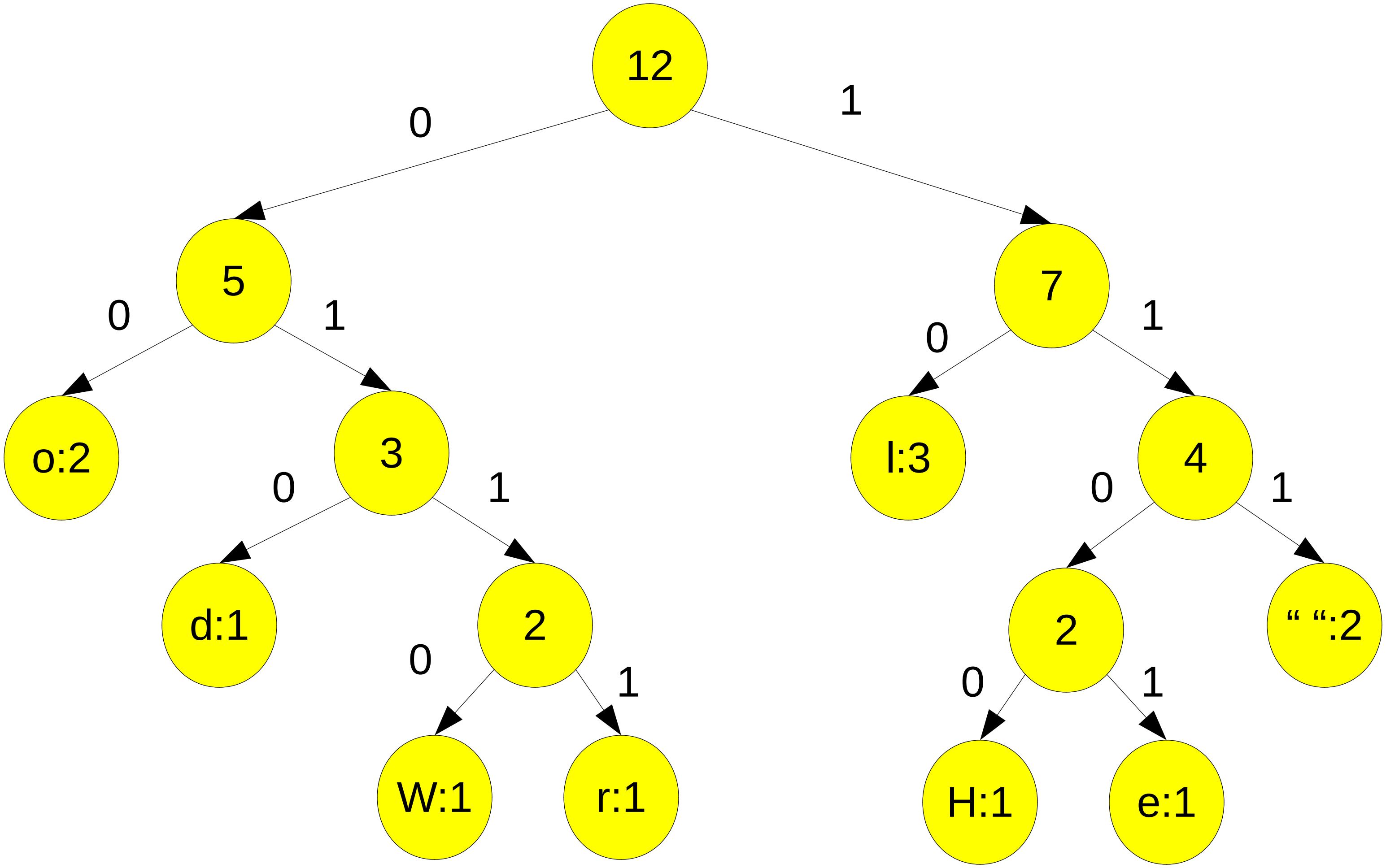
What is the code for e?



1101

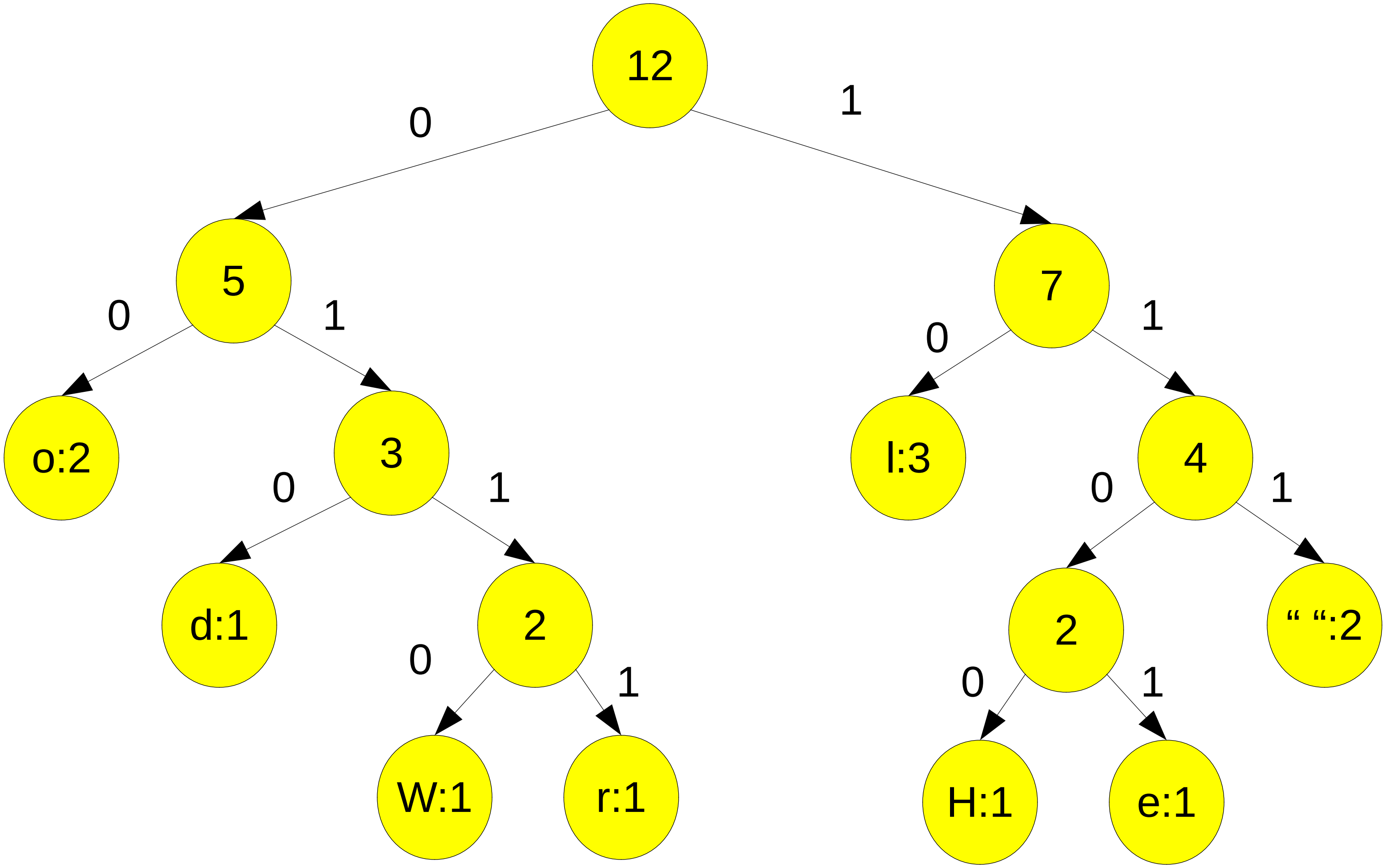
Why we do this

- We merged the least frequent characters first so they will be deeper in the end tree, so have a longer encoding. So the more frequent characters are closer to the top.
- Our encoding is prefix-free since we'd have to traverse past a leaf node to encode a prefix.



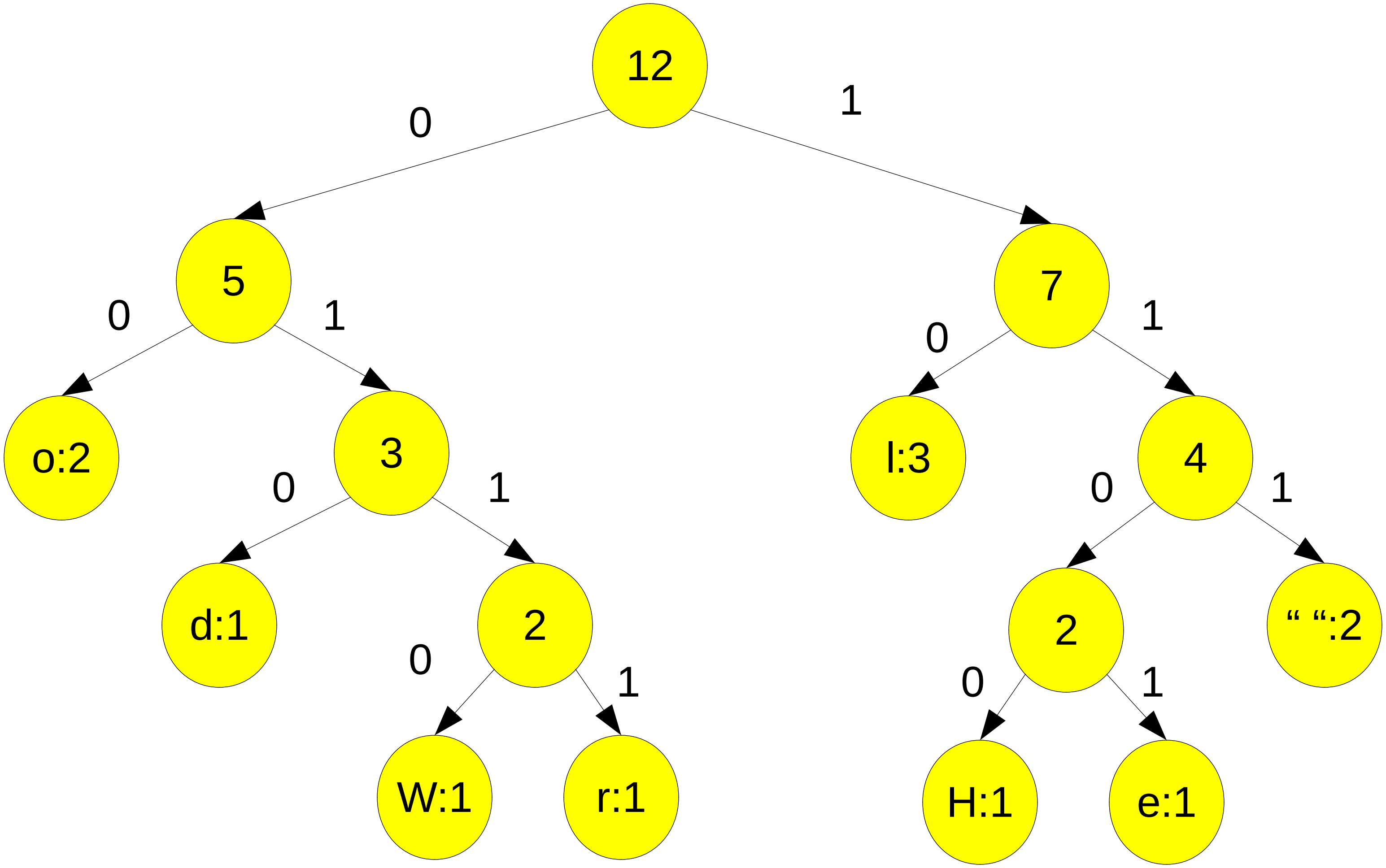
Encoding for “Hello World “
1100 1101 10 10 00 111 0110 00 0111 10 010 111
H e l l o _ W o r l d _

35 bits vs 36 for a 3-bit encoding, or 96 for ASCII or 192 for Unicode



How would we encode Hole?

1100 00 10 1101



Decode: 10110100

0111110111010101110100000111

Compression Note

- There is no algorithm that will always achieve a smaller file which we can decompress.
  - Otherwise we could keep compressing until a file is 1 bit, or even 0 bits.
- In the case of Huffman Coding, if we are using this to compress a file, we need to store the tree along with the encoding, which means there is always some overhead.