CSCI 2270

Data Structures & Algorithms

Gabe Johnson

Lecture 26 Mar 18, 2013

Review for Exam 2

Upcoming Homework Assignment

HW #7 **Due: Mon, Mar 18**

Huffman Encoding

Due later today, 6pm

Lecture Goals

1. Review w/ Alec: Tues 3pm in the Fishbowl
2. Review for Test (Wednesday)

Test Format

Same as last time:

50 minutes
Open Note
Closed Gizmo

Take a test from the front at the beginning of class, but keep it face down until I say ‘go’.

Topics

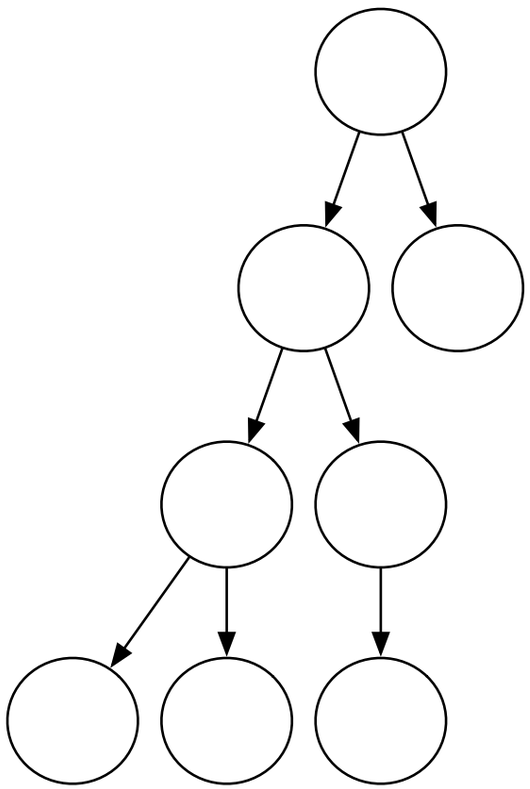
Recursion
Pointers
Computational Complexity
Binary Trees
B-Trees
Priority Queues
Huffman Encoding

Very Little Coding

Last time you wrote a lot of code. This time you won't. You will be expected to read code, apply algorithms, and report the outcome of those algorithms.

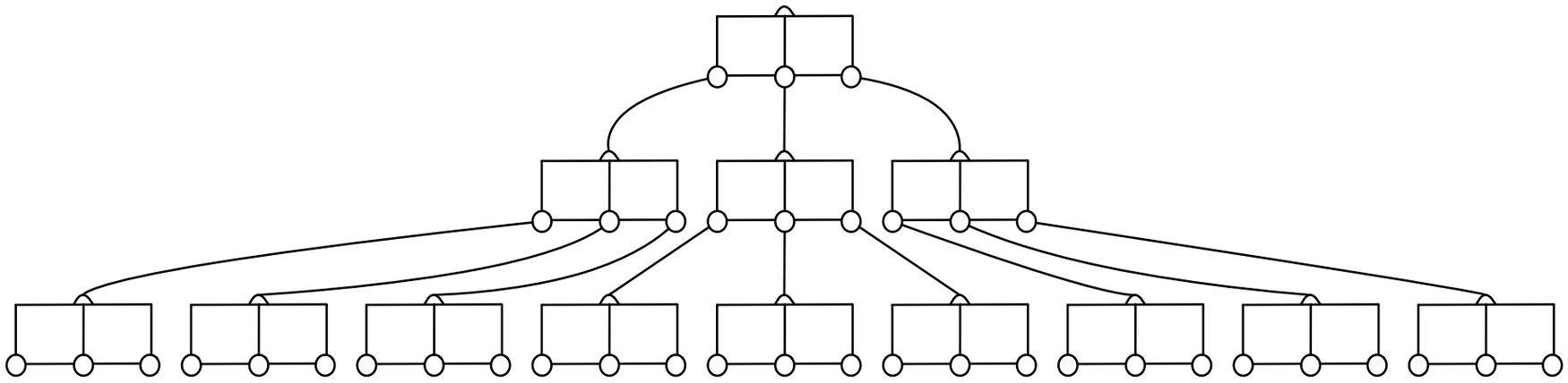
No Evil B-Tree Question

There is one B-Tree question and it is about computational complexity.



The runtime complexity of many operations in *binary* trees involves the base-2 logarithm of the input size *n*. This is because every time we follow an edge to a child, we are reducing our search space by *half*.

B-Tree Complexity

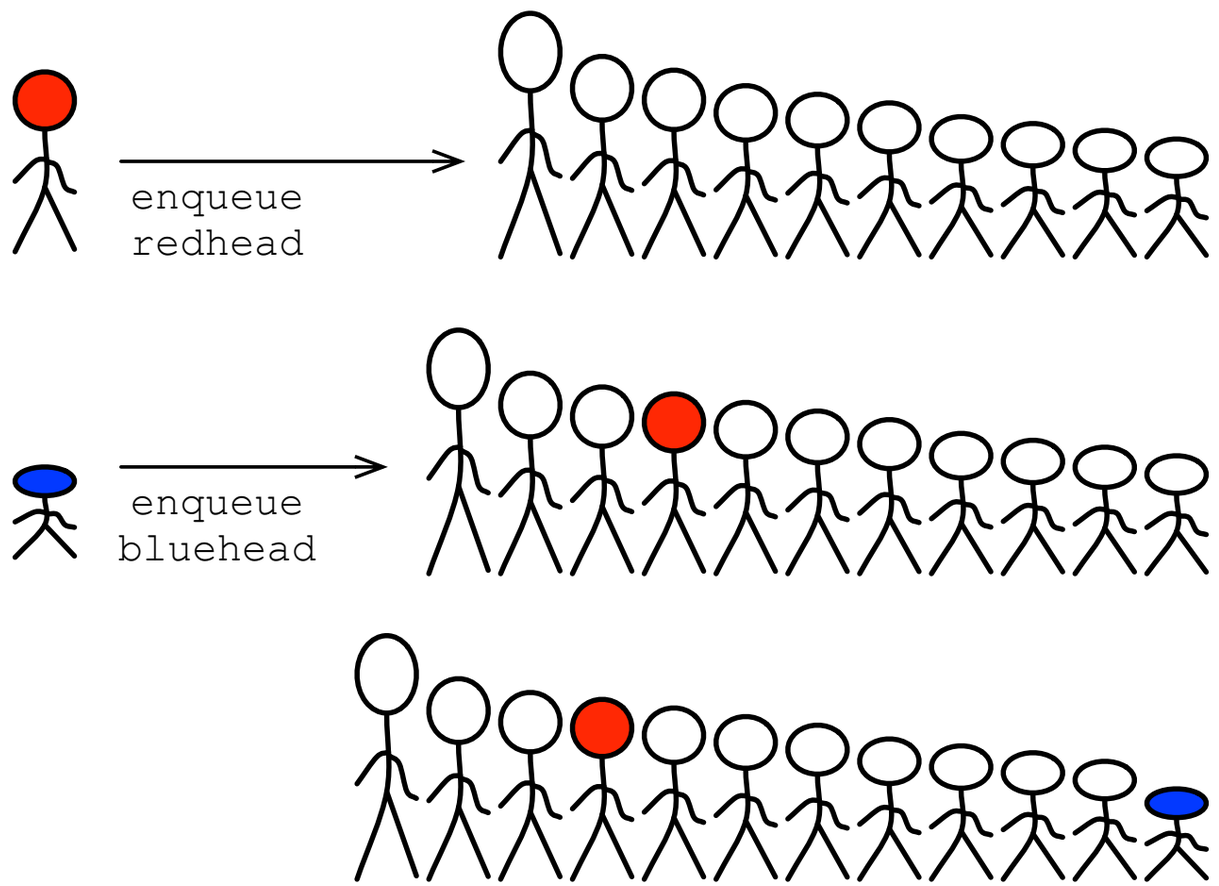


An order *m* B-Tree has *m* children per non-leaf node. This means that when we follow a child link, we reduce the search space by a factor of *m*. This means that for B-Trees, we'll have big-oh complexity involving $\mathbf{\log_m n}$, while binary trees (2 children per node) involved $\mathbf{\log_2 n}$ complexity.

Priority Queues

A priority queue maintains an order of how items are removed. We can use any sortable data for the priority: alphabetizing, integers, floating point numbers, etc.

In this silly example we add people to a priority queue that arranges people by height. The highest priority person is the first to be removed, regardless of when they arrived.



Priority Queue Operations

There are three classical operations for queues (including priority queues). Depending on the language they have different names and sometimes different semantics. But there are always at least these:

add / insert / enqueue: add an item to the PQ.

peek / top: get the highest priority element but don't remove it.

pop / remove / dequeue: erase the highest priority element. usually this returns it, but C++ is cranky and doesn't.

Huffman

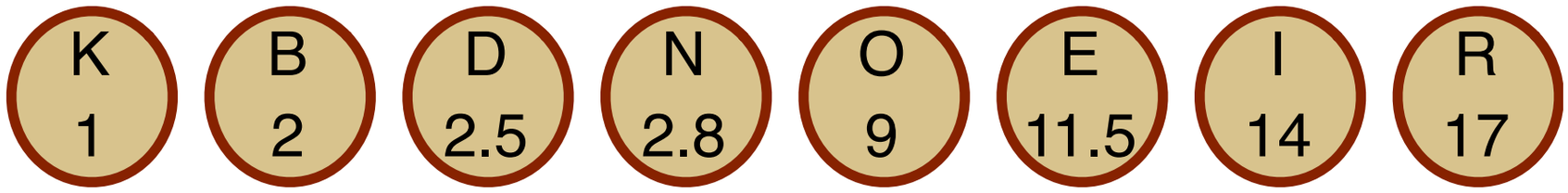
I must have Huffman Encoding on my mind, because a giant chunk of the test concerns it.

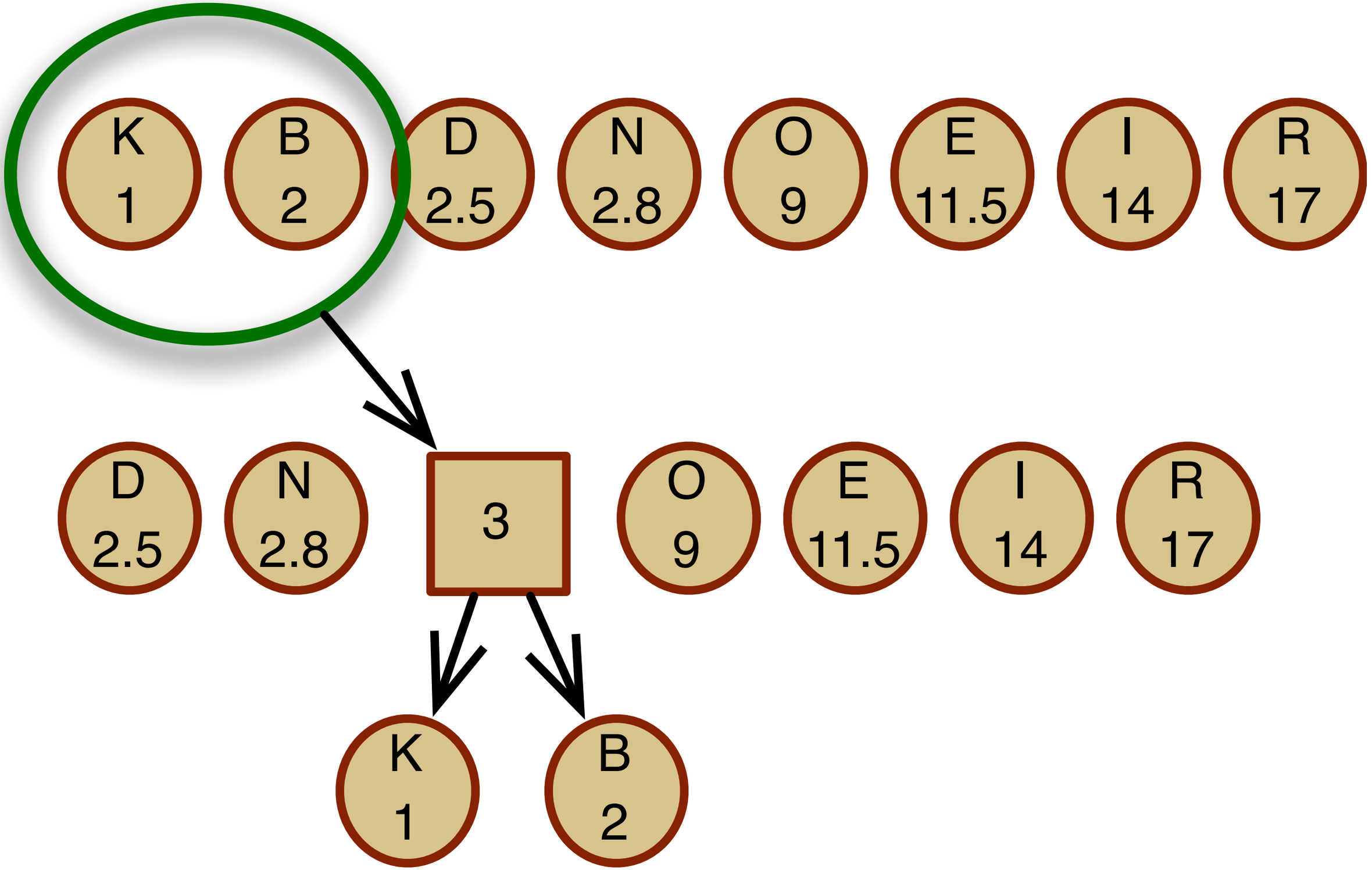
You won't need to write code. Instead, I'll give you an initial condition, and ask you to apply various algorithms. This is very diagram-oriented.

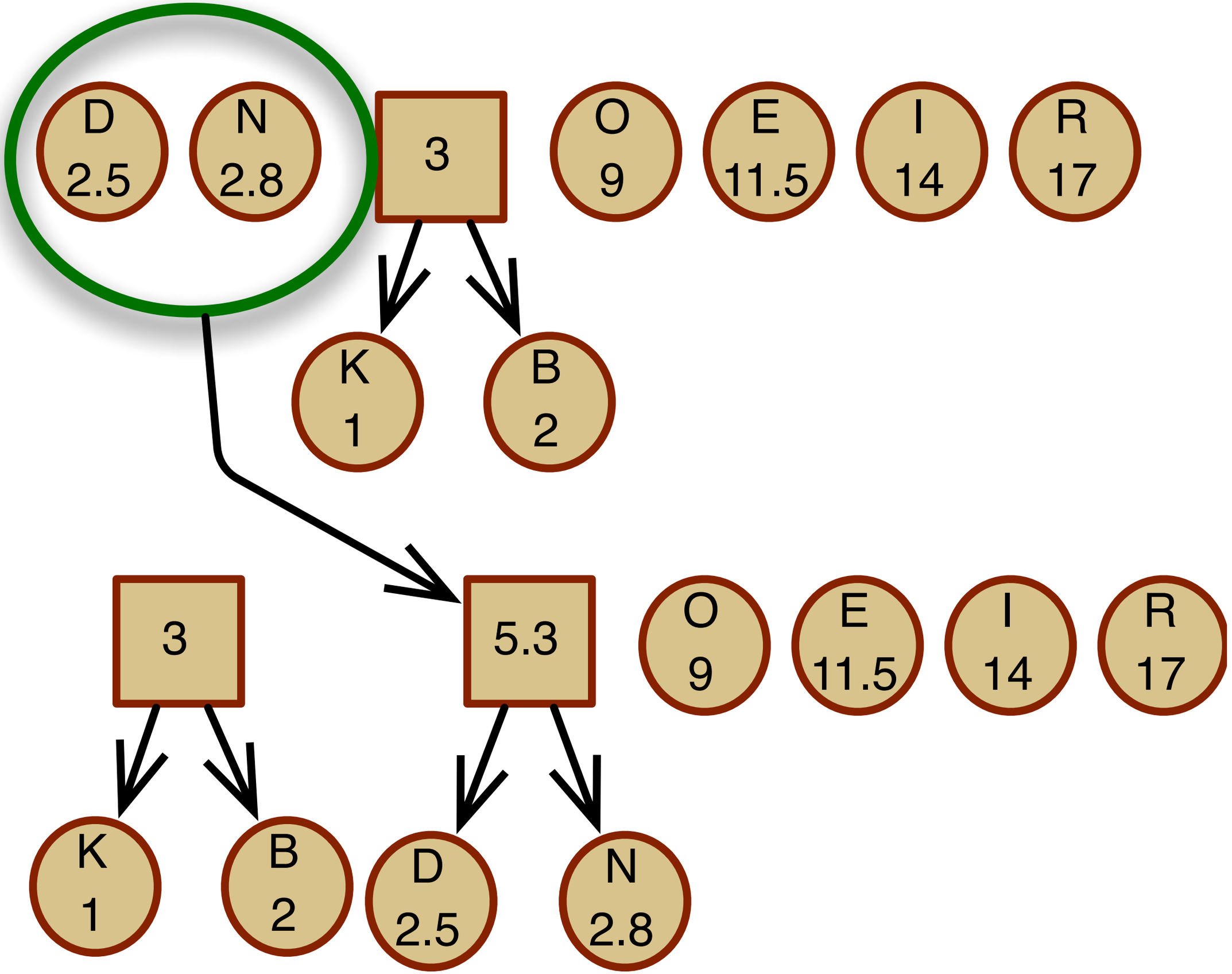
So, let's look at some diagrams.

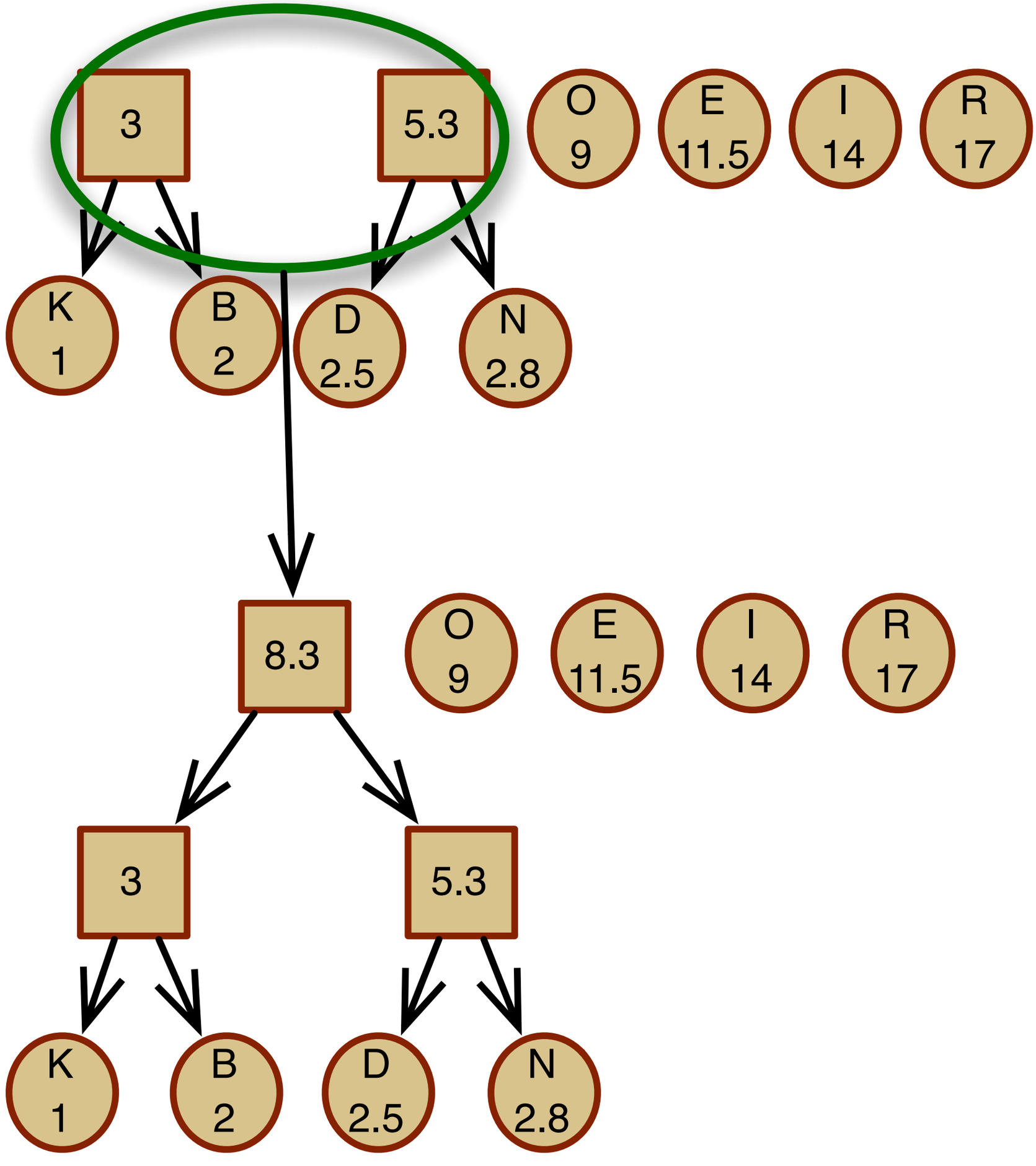
1. Create Tree

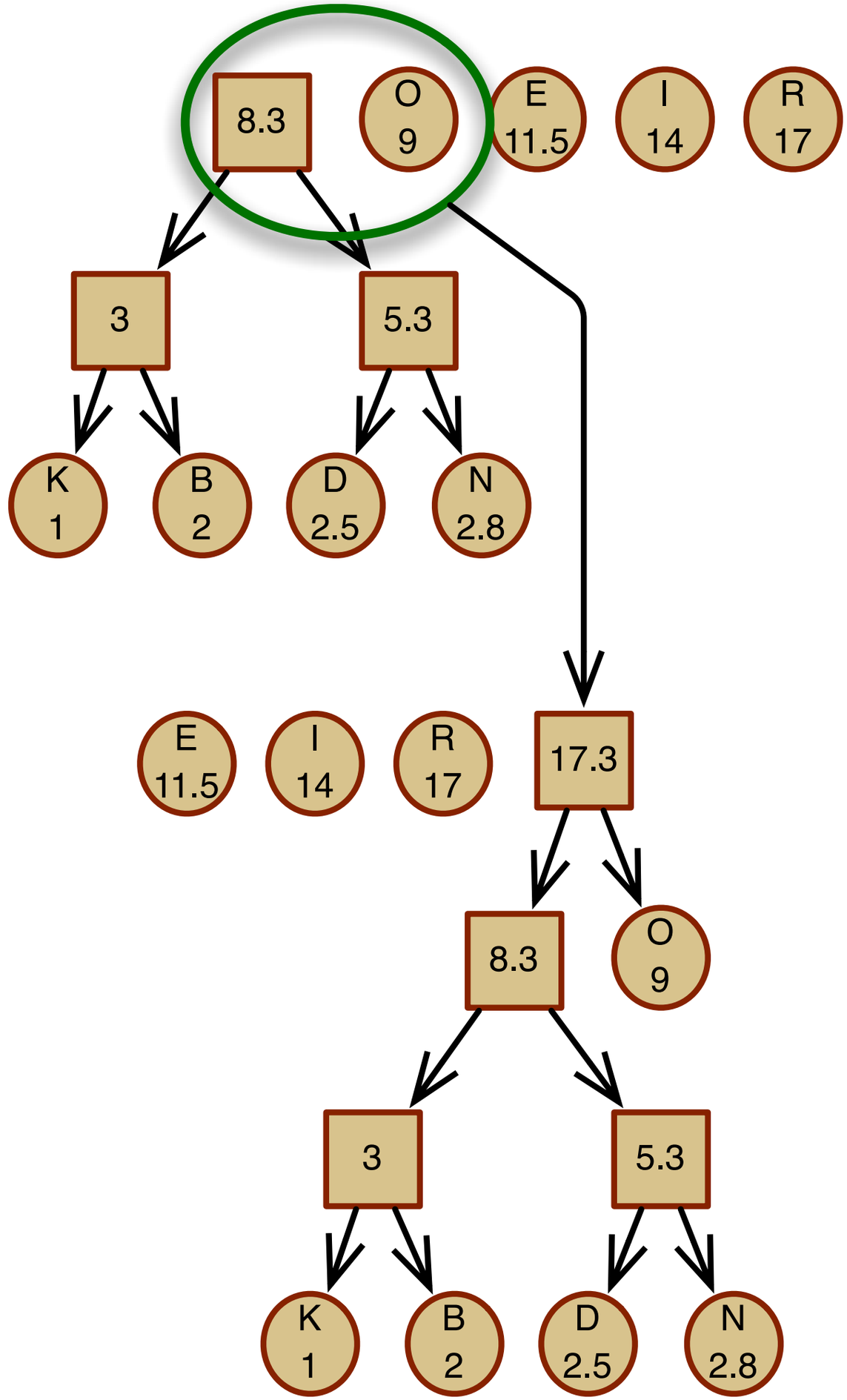
I'll give you an initial priority queue with symbols and frequencies and ask you to run the algorithm that generates a tree. Something like this:

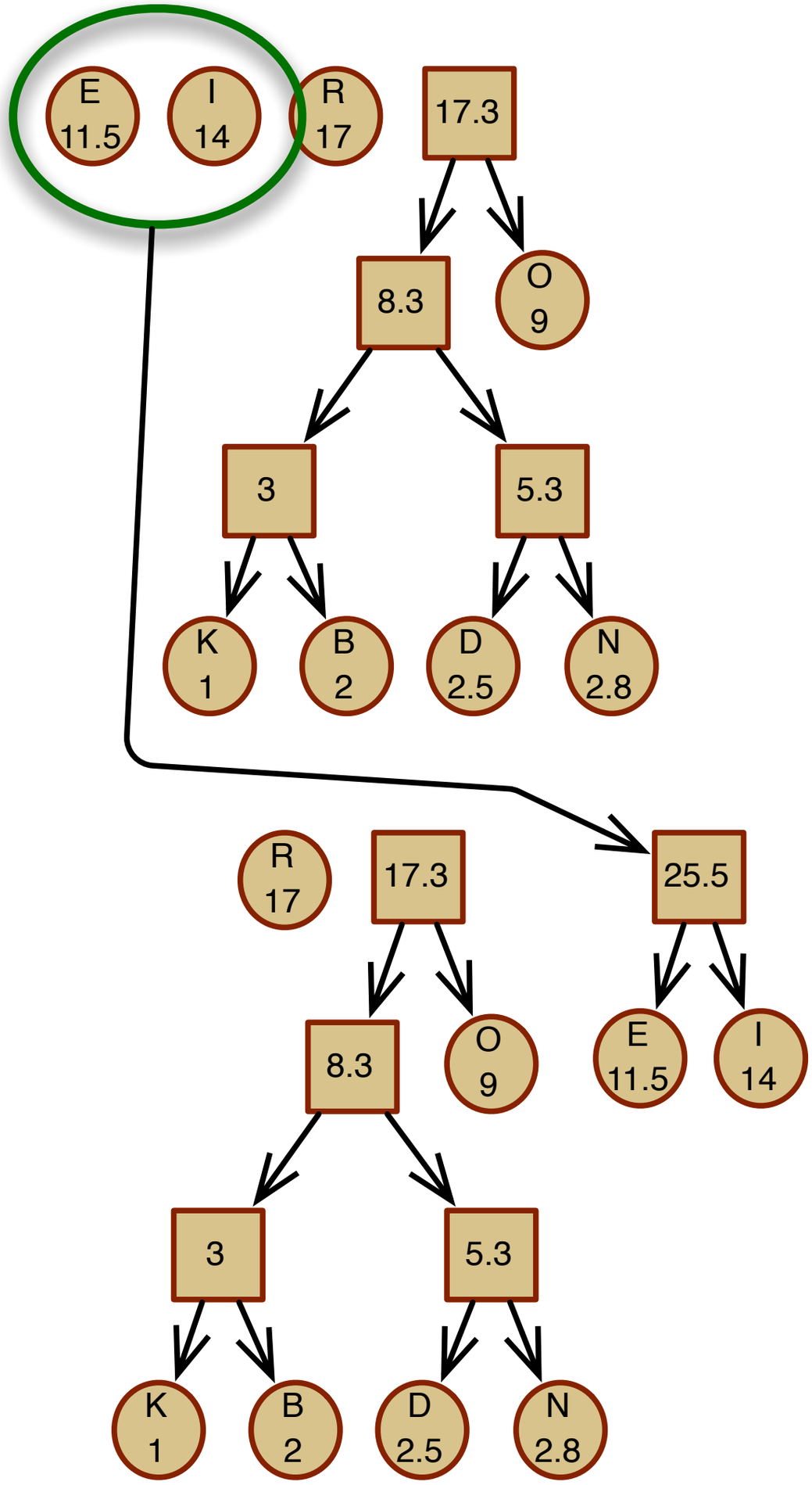


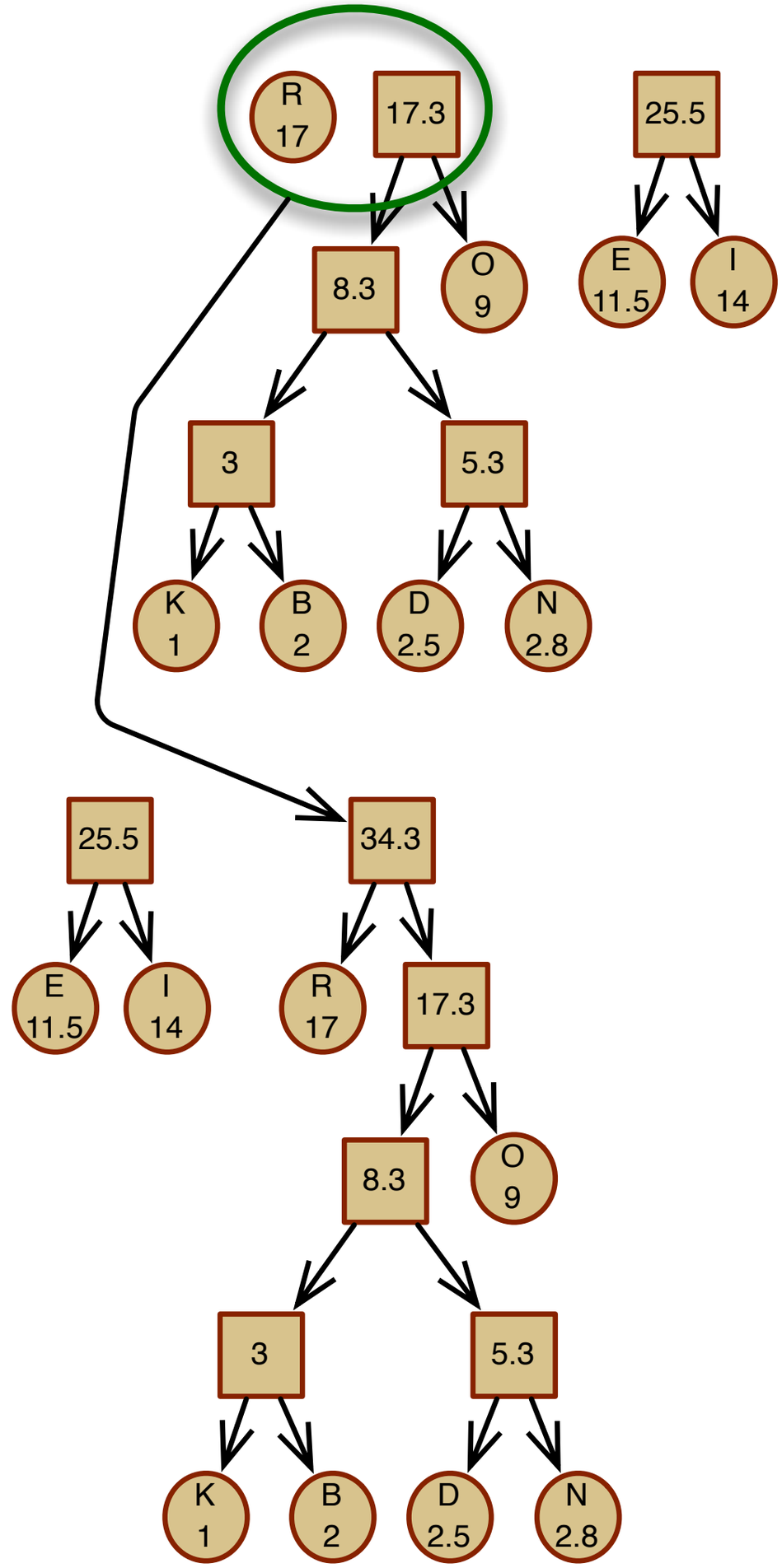

K
1
B
2
D
2.5
N
2.8
O
9
E
11.5
I
14
R
17
D
2.5
N
2.8
3
O
9
E
11.5
I
14
R
17
K
1
B
2

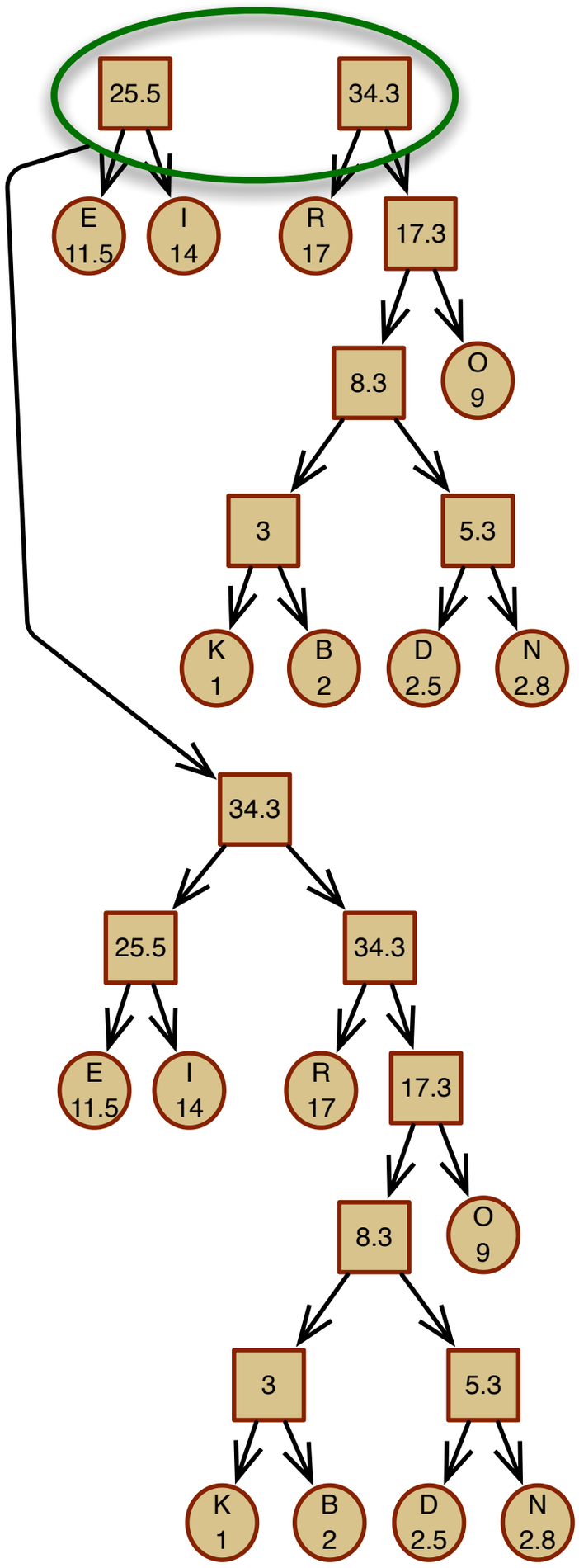

D
2.5
N
2.8
3
O
9
E
11.5
I
14
R
17
K
1
B
2
3
5.3
O
9
E
11.5
I
14
R
17
K
1
B
2
D
2.5
N
2.8


3
5.3
O
9
E
11.5
I
14
R
17
K
1
B
2
D
2.5
N
2.8
8.3
O
9
E
11.5
I
14
R
17
3
5.3
K
1
B
2
D
2.5
N
2.8


8.3
O
9
E
11.5
I
14
R
17
3
5.3
K
1
B
2
D
2.5
N
2.8
E
11.5
I
14
R
17
17.3
8.3
O
9
3
5.3
K
1
B
2
D
2.5
N
2.8


E
11.5
I
14
R
17
17.3
8.3
O
9
3
5.3
K
1
B
2
D
2.5
N
2.8
25.5
R
17
17.3
8.3
O
9
E
11.5
I
14
3
5.3
K
1
B
2
D
2.5
N
2.8


R
17
17.3
25.5
8.3
O
9
E
11.5
I
14
3
5.3
K
1
B
2
D
2.5
N
2.8
25.5
34.3
E
11.5
I
14
R
17
17.3
8.3
O
9
3
5.3
K
1
B
2
D
2.5
N
2.8

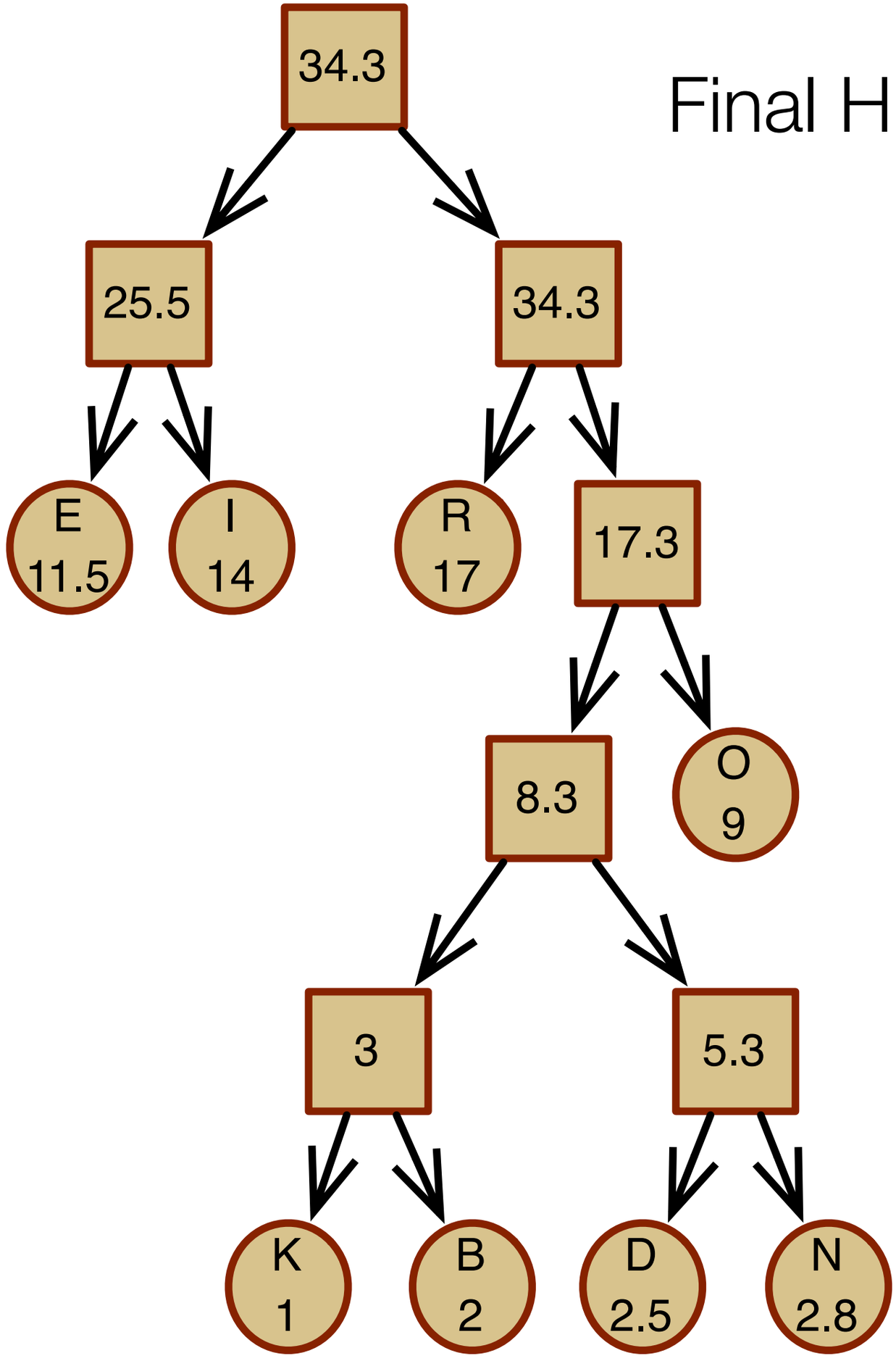


This is the last step in combining nodes from the priority queue. We are left with a single tree. Internal nodes hold accumulated frequency data.

Leaf nodes hold symbol data and their original frequency data.

The path from root to leaf node specifies the encoding for the node's symbol. We commonly represent left traversal as 0 and right traversal as 1.

Final Huffman Tree



Then

Using that encoding tree:

* Be able to make a symbol lookup table that relates letters to bitstrings. E.g. **`K = 11000.`**

* Be able to decode bitstrings based on the encoding tree. E.g. **`1011111011 = RON`**