38.2 Prefix-free Codes

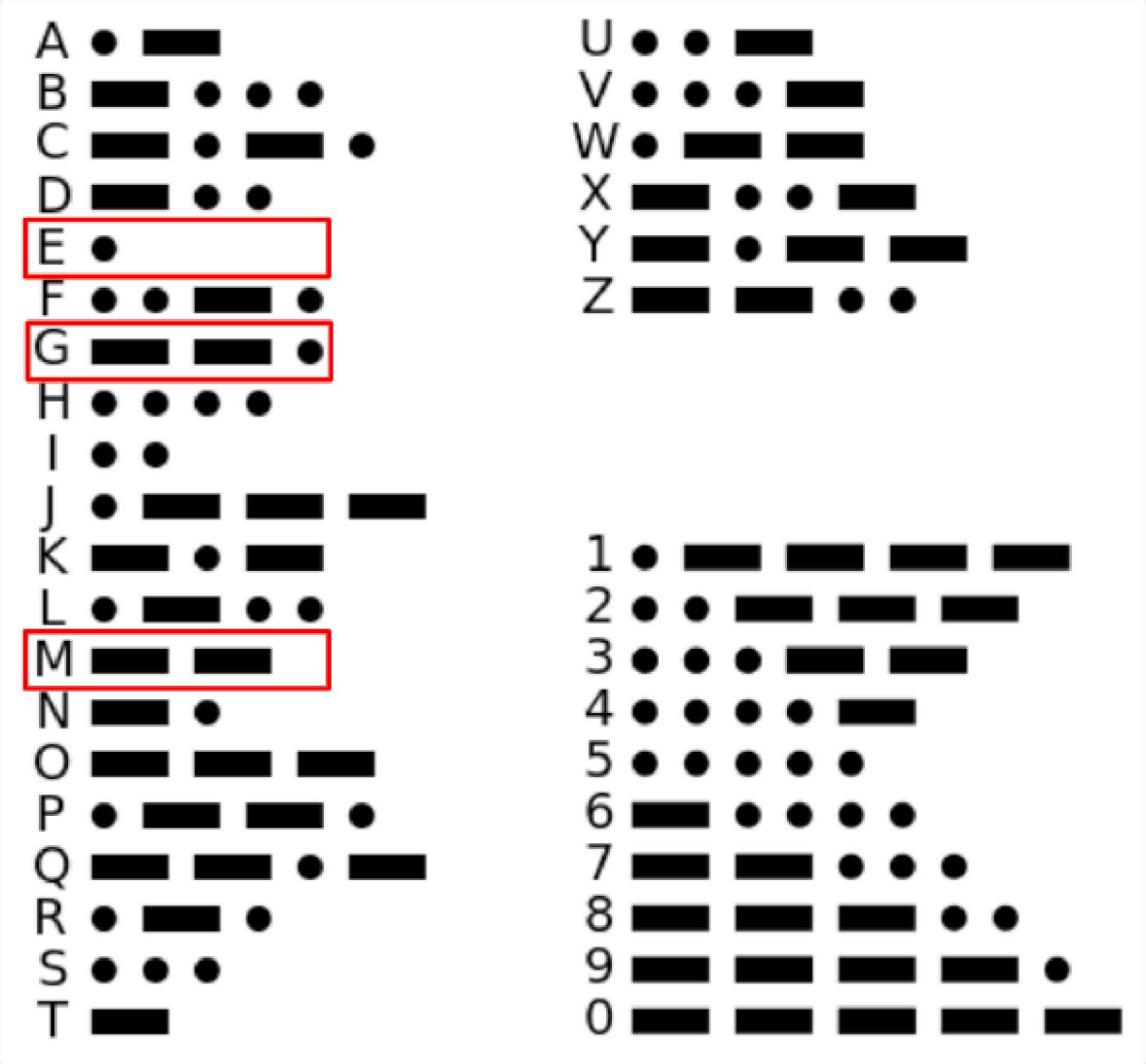
Consider the representation of English text in Java. We represent text as a sequence of characters, each taking 8 bits of memory.

One easy way to compress, then, is to simply use less than 8 bits per character. To do this, we have to decide which **codewords** (bit sequences) go with each **symbol** (character).

Mapping Alphanumeric Symbols

Morse Code

As an introductory example, consider the Morse code alphabet. Looking at the alphabet below, what does the sequence – – • – – • represent? It's ambiguous! The same sequence of symbols can represent either MEME, or GG, depending on what you choose – – • to represent



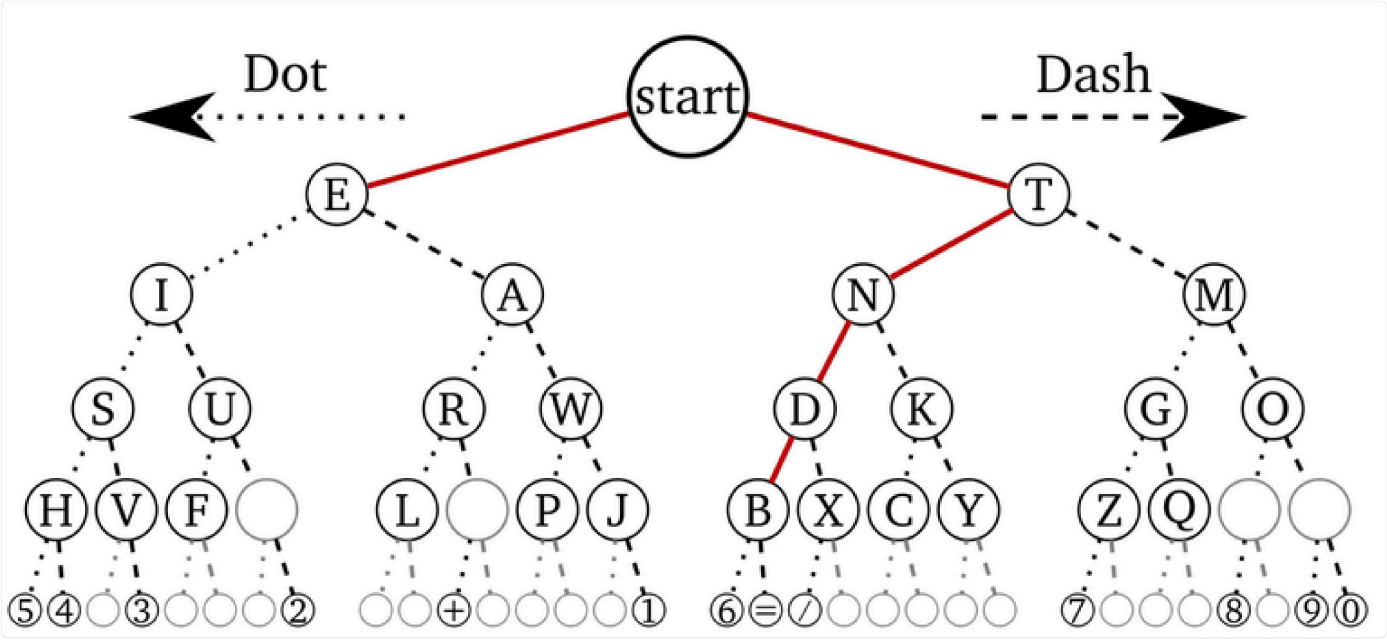
Ambiguity in morse code

In real usage, operators must pause between codewords to indicate a break. The pause acts as an implicit third symbol, but we can't encode this real-time information into our code.

Prefix-free Codes

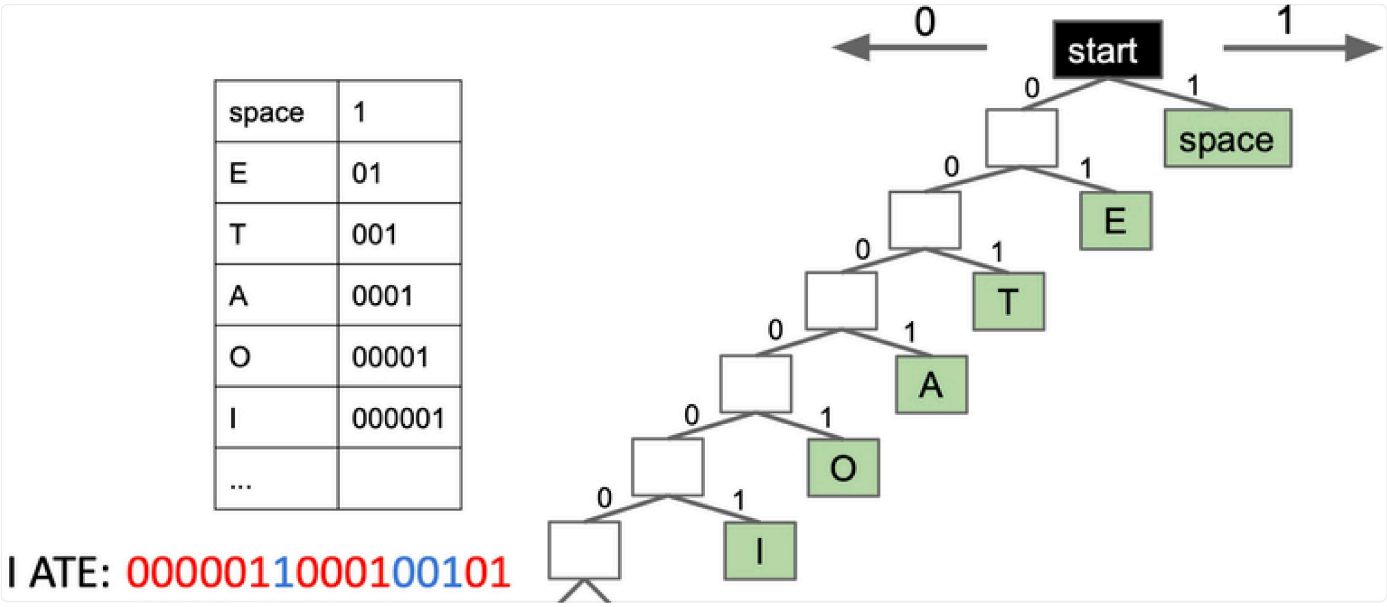
An alternate strategy to avoid the need for real-time is to use **prefix-free codes**. In a prefix-free code, no codeword is a prefix of any other. In the Morse Code example, there would be no confusion whether the – – in the pattern – – • – – • is supposed to represent M, or the start of G.

Let's represent Morse code as a tree of codewords leading to symbols. As we can see from the tree, several symbols have representations that are prefixes of other symbols.



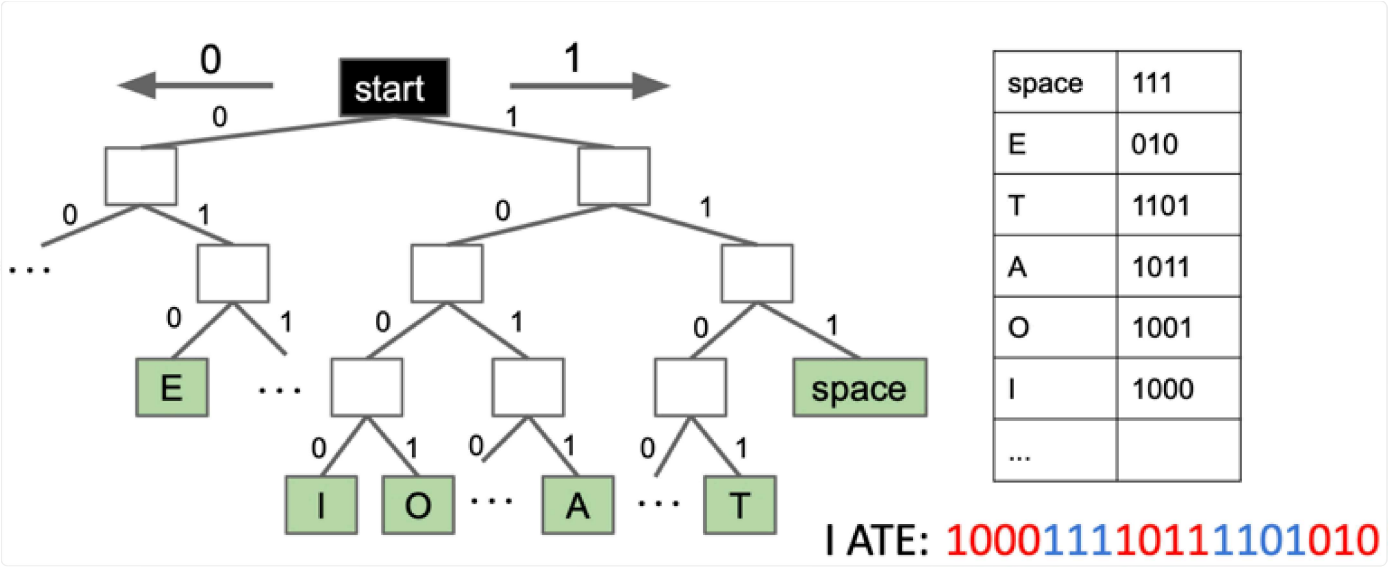
Morse code is not prefix-free.

As an example of an (arbitrary) prefix-free code, consider the following encoding:



One prefix-free code.

The following code is also prefix-free:



space	111
E	010
T	1101
A	1011
O	1001
I	1000
...	

Another prefix-free code.

Note that some codes are more efficient for certain strings than others: in the first representation, `I ATE` uses less bits than the second code. However, this is highly dependent on what string we're trying to encode.

Previous
38.1 Introduction to Compression

Next
38.3 Shannon-Fano Codes

Last updated 1 year ago