```
To encode using the DMC method,
  1. Set $s \leftarrow 1$. /* the current number of states */
  2. Set $t \leftarrow 1$. /* the current state */
  3. Set $T[1][0] \leftarrow 1$ and $T[1][1] \leftarrow 1$. /* initial model */
  4. Set $C[1][0] \leftarrow 1$ and $C[1][1] \leftarrow 1$. /* set counts to 1 to avoi
     zero-frequency problem */
  5. For each input bit e do
     (a) Set $u \leftarrow t$.
     (b) Set $t \leftarrow T[u][e]$. /* follow the transition */
     (c) Code e with probability $C[u][e]/(C[u][0] + C[u][1])$.
     (d) Set $C[u][e] \leftarrow C[u][e] + 1$.
     (e) If cloning thresholds are exceeded then
            Set $s \leftarrow s + 1$ /* the new state ($t'$) */,
            Set $T[u][e] \leftarrow s$,
            Set $T[s][0] \leftarrow T[t][0]$,
            Set $T[s][1] \leftarrow T[t][1]$, and
            Move some of the counts from $C[t]$ to $C[s]$.
```

Figure 2.31 Encoding using DMC.

ually, and this can be slow. Bitwise DMC is probably one of the easiest models to implement—only a few lines of code are needed to perform the counting and cloning—yet DMC is one of the best compression methods in terms of compression performance. Its speed can be improved by adapting it to work with bytes rather than bits, but this requires more sophisticated data structures to avoid excessive memory usage. In this case, the implementation effort becomes comparable to that of other high-performance methods, such as PPM, and the difference between the two methods—both in implementation effort and compression efficiency—becomes small.

Word-based compression

Word-based compression methods parse a document into "words" (typically, contiguous alphanumeric characters) and "nonwords" (typically, punctuation and white-space characters) between the words. The words and nonwords become the symbols to be compressed. There are various ways to compress them. Generally, the most effective approach is to form a zero-order model for words and another for nonwords. It is assumed that the text consists of strictly alternating words and nonwords (the parsing method needs to ensure this), and so the two models are used alternately. If the models are adaptive, a means of transmitting previously unseen words and nonwords is required. Usually, some escape symbol is transmitted, and then the novel word is spelled out character by character. The explicit characters can be compressed using a simple model, typically a zero-order model of the characters.

Although this approach seems to be specific to textual documents, it does not perform too badly on other types of data. This is because if few "words" are found, then the model effectively reverts to the simple zero-order model of characters, and for nontextual data such as images, this sort of model is reasonably appropriate. A well-tuned word-based compressor can achieve compression performance close to that of PPM, and it has the potential to be substantially faster because it codes several characters at a time.

There are many different ways to break English text into words and the intervening nonwords.[3] One scheme is to treat any string of contiguous alphabetic characters as a word and anything else as a nonword. More sophisticated schemes could take into account punctuation that is part of a word, such as apostrophes and hyphens, and even accommodate some likely sequences, such as a capital letter following a period. This kind of improvement does not have much effect on compression but may make the resulting list of words more useful for indexing purposes in a full-text retrieval system.

One aspect of parsing that deserves attention is the processing of numbers. If digits are treated in the same way as letters, a sequence of digits will be parsed as a word. This can cause problems if a document contains many numbers—such as tables of financial figures. The same situation occurs, and can easily be overlooked, when a large document contains page numbers—with 100,000 pages, the page numbers will generate 100,000 "words," each of which occurs only once. Such a host of unique words can have a serious impact on operation: in an adaptive system, each one must be spelled out explicitly, and in a static system, each will be stored in the compression model. In both cases this is grossly inefficient because the frequency distribution of these numbers is quite different from the frequency distribution of normal words for which the system is designed. One solution is to limit the length of numbers to just a few digits. Longer numbers are broken up into shorter ones, with a null punctuation marker in between. This moderates the vocabulary generated by the numbers and can result in considerable savings in decode-time memory requirements.

Word-based schemes can generate a large number of symbols since there is a different symbol for each word and word variant that appears in the text being coded. This means that special attention must be given to efficient data structures for storing the model. In a static or semi-static situation, a canonical Huffman code is

[3] Note, however, that not all languages are so cooperative. Most Asian languages are written and stored without any equivalent of the white-space characters that are taken for granted in English, and they are very difficult to segment into words. For compression purposes, this is not a serious handicap, as other methods can be used. But for information retrieval purposes (see Chapter 4), where discrete words are employed as the terms used to search for documents relevant to a query, segmentation is an important problem indeed.