The spelling correction problem however demands more than computing edit distance: given a set $\mathcal{S}$ of strings (corresponding to terms in the vocabulary) and a query string q, we seek the string(s) in V of least edit distance from q. We may view this as a decoding problem, in which the codewords (the strings in V) are prescribed in advance. The obvious way of doing this is to compute the edit distance from q to each string in V, before selecting the string(s) of minimum edit distance. This exhaustive search is inordinately expensive. Accordingly, a number of heuristics are used in practice to efficiently retrieve vocabulary terms likely to have low edit distance to the query term(s).

The simplest such heuristic is to restrict the search to dictionary terms beginning with the same letter as the query string; the hope is that spelling errors do not occur in the first character of the query. A more sophisticated variant of this heuristic is to use a version of the permuterm index, in which we omit the end-of-word symbol \$. Consider the set of all rotations of the query string q. For each rotation r from this set, we traverse the B-tree into the permuterm index, thereby retrieving all dictionary terms that have a rotation beginning with r. For instance, if q is mase and we consider the rotation $r =$ sema, we would retrieve dictionary terms such as semantic and semaphore, which do not have a small edit distance to q. Unfortunately, we would miss more pertinent dictionary terms such as mare and mane. To address this, we refine this rotation scheme: for each rotation, we omit a suffix of ℓ characters before performing the B-tree traversal. This ensures that each term in the set R of terms retrieved from the dictionary includes a "long" substring in common with q. The value of ℓ could depend on the length of q. Alternatively, we may set it to a fixed constant such as 2.

3.3.4 k-Gram indexes for spelling correction

To further limit the set of vocabulary terms for which we compute edit distances to the query term, we now show how to invoke the k-gram index of Section 3.2.2 (page 50) to assist with retrieving vocabulary terms with low edit distance to the query q. Once we retrieve such terms, we can then find the ones of least edit distance from q.

In fact, we use the k-gram index to retrieve vocabulary terms that have many k-grams in common with the query. We argue that, for reasonable definitions of "many k-grams in common," the retrieval process is essentially that of a single scan through the postings for the k-grams in the query string q.

The 2-gram (or *bigram*) index in Figure 3.7 shows (a portion of) the postings for the three bigrams in the query bord. Suppose we wanted to retrieve vocabulary terms that contained at least two of these three bigrams. A single scan of the postings (much as in Chapter 1) would let us enumerate all such

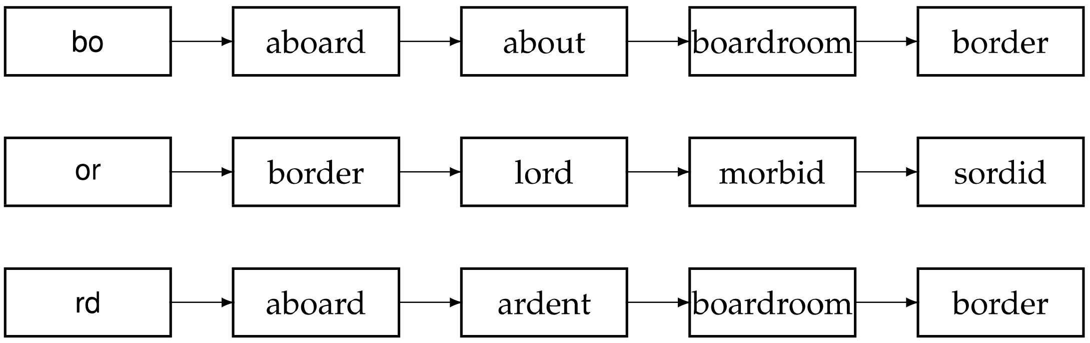

Figure 3.7 Matching at least two of the three 2-grams in the query bord.

terms; in the example of Figure 3.7, we would enumerate aboard, boardroom, and border.

This straightforward application of the linear scan intersection of postings immediately reveals the shortcoming of simply requiring matched vocabulary terms to contain a fixed number of k-grams from the query q: terms like boardroom, an implausible "correction" of bord, get enumerated. Consequently, we require more nuanced measures of the overlap in k-grams between a vocabulary term and q. The linear scan intersection can be adapted when the measure of overlap is the *Jaccard coefficient* for measuring the overlap between two sets A and B, defined to be $|A \cap B|/|A \cup B|$. The two sets we consider are the set of k-grams in the query q, and the set of k-grams in a vocabulary term. As the scan proceeds, we proceed from one vocabulary term t to the next, computing on the fly the Jaccard coefficient between q and t. If the coefficient exceeds a preset threshold, we add t to the output; if not, we move on to the next term in the postings. To compute the Jaccard coefficient, we need the set of k-grams in q and t.

JACCARD COEFFICIENT

Because we are scanning the postings for all k-grams in q, we immediately have these k-grams on hand. What about the k-grams of t? In principle, we could enumerate these on the fly from t. In practice, this is not only slow but potentially infeasible; in all likelihood, the postings entries themselves do not contain the complete string t but rather some encoding of t. The crucial observation is that to compute the Jaccard coefficient, we only need the length of the string t. To see this, recall the example of Figure 3.7 and consider the point when the postings scan for query $q =$ bord reaches term $t =$ boardroom. We know that two bigrams match. If the postings stored the (precomputed) number of bigrams in boardroom (namely, 8), we have all the information we require to compute the Jaccard coefficient to be $2/(8 + 3 - 2)$; the numerator is obtained from the number of postings hits (2, from bo and rd); the denominator is the sum of the number of bigrams in bord and boardroom, less the number of postings hits.

We could replace the Jaccard coefficient by other measures that allow efficient on the fly computation during postings scans. How do we use these for spelling correction? One method that has some empirical support is to first use the k-gram index to enumerate a set of candidate vocabulary terms