benefitted from the criticisms of Carl Smith, Doug Comer, Dirk Siefkes, and Fran Berman.

Received 4/80; revised 12/81; accepted 3/82

References
5. Clarke, E.M. Programming language constructs for which it is impossible to obtain good Hoare-like axiom systems, JACM 26, 1 (Jan. 1979), 129-147.

A Hash Code Method for Detecting and Correcting Spelling Errors

M. Mor and A. S. Fraenkel
The Weizmann Institute of Science, Israel

The most common spelling errors are one extra letter, one missing letter, one wrong letter, or the transposition of two letters. Deletion, exchange, and rotation operators are defined which detect and "mend" such spelling errors and thus permit retrieval despite the errors. These three operators essentially delete a letter of a word, exchange two adjacent letters, and rotate a word cyclically. Moreover, the operators can be used in conjunction with hashing, thus permitting very fast retrieval. Results of experiments run on large databases in Hebrew and in English are briefly indicated.

CR Categories and Subject Descriptors: E.2 [Data Storage Representations]—Hash-table representations; H.3.3, [Information Storage and Retrieval]: Content Analysis and Indexing—dictionaries; H.3.3: [Information Storage and Retrieval]: Information Search and Retrieval—search process

General Terms: Algorithms, Experimentation, Theory

Additional Key Words and Phrases: spelling, spelling errors, deletion, exchange, rotation, dictionary

Introduction

In a recent survey on detecting and correcting spelling errors, Peterson [4] states that for locating words in a word list, "...a hash table approach would seem the most reasonable." However, all the algorithms described do not use hashing or use it only to restrict the search range (e.g., the cited DEC-10 SPELL program) in which
case the hashing is followed by a sequential or binary search. Peterson then cites Damerau [2] who wrote that an inspection showed that over 80 percent of all spelling errors fall into one of the following four categories of single errors: (1) one extra letter (insertion); (2) one missing letter (deletion); (3) one wrong letter (substitution); (4) transposition of two adjacent letters (transposition). We call these single errors. Peterson also adds that deletion and substitution are difficult to detect.

We have developed a method for overcoming single errors by using special operators that operate on the so-called significant words, that is, the textwords that are deemed sufficiently important to serve as keywords in queries. (In a full-text retrieval system, the significant words may be all the textwords or all but some 150 common words.) The method is implemented by means of hashing, and no other table look-ups are needed. However, it may be implemented by other search techniques, such as binary search in a sorted table, instead of hashing.

The idea behind the main variation of the method is to store every significant word \( x \) in a hash table \( |x| + 1 \) times, where \( |x| \) denotes the length of \( x \), each time omitting one of the letters. Also, \( x \) itself is stored. Suppose that a textword \( w \) is a single-error occurrence of \( x \). If one of the letters of \( x \) is missing in \( w \), it should be clear that precisely one of the \( |x| \) stored forms of \( x \) with one letter omitted will retrieve \( w \). If \( w \) has an extra letter, then operating on \( w \) with deletions as we did on \( x \) will retrieve \( w \) via \( x \) itself. Substitutions and transpositions are handled similarly to deletion.

The average number of table accesses is \( O(|w|) \), where the implied constant depends on the hash function and the number of distinct operators (at most three). The operators are defined in the next section which also contains a description of the hash table construction and the search algorithm for four variations of the method. We also indicate how to extend our approach to detect some errors other than single errors.

The method was tested on large Hebrew and English databases. Results of these tests as well as a partial literature survey of approaches to solve the problem of correcting spelling errors are given elsewhere [3]. The tests confirm the effectiveness of the method by showing that it does not generate excessive noise and is indeed capable of improving the recall of searches.

This effectiveness is achieved at the price of extra memory space. However, since memories tend to get larger and cheaper at a rather fast pace, the method's practicality should increase with time even for very large dictionaries. Moreover, our method increases only the size of the dictionary of significant words. In many full-text retrieval systems the dictionary of all distinct words is only a small part of the entire memory requirement. In such systems the dictionary points to a concordance, which is an inverted list that contains for every dictionary word all its locations in the text. Retrieval is performed using only the dictionary and concordance.

Note that the retrieval of a word \( w \) is symmetric with respect to the error: either \( w \) contains a single error and the dictionary word \( x \) is correct or vice versa (but we assume that \( w \) and \( x \) are not both erroneous). To concretize the discussion we assume that \( w \) contains a single error but \( x \) is correct.

### Description of the Method

The method is based on using combinations of deletion operators \( D_i \), exchange operators \( E_i \), and rotation operators \( R_i \) and hence is called the RED method. These operators operate on the significant words and on the words searched for as detailed below. Given a word \( y = y_1 y_2 \ldots y_{t-1} y_t \) consisting of \( t \) characters \( y_1, \ldots, y_t \), the RED operators are defined as follows:

\[
D_i(y) = y_1 y_2 \ldots y_{i-1} y_{i+1} \ldots y_t \quad (1 \leq i \leq t)
\]

that is, deletion of the \( i \)th character of \( y \). The set of all deletions of one character of \( y \) is denoted by \( D = D(y) = \{D_1, \ldots, D_t\} \). A typical exchange operator is

\[
E_i(y) = y_1 y_2 \ldots y_{i-1} y_{i+1} y_i y_{i+2} \ldots y_t \quad (1 \leq i \leq t-1)
\]

that is, transposition of two adjacent characters in \( y \). The set of all adjacent transpositions of \( y \) is denoted by \( E \). Finally, a typical rotation operator is

\[
R_i(y) = y_{i+1} y_{i+2} \ldots y_t y_1 \ldots y_i \quad (1 \leq i \leq t)
\]

that is, rotation of \( y \) by \( i \) positions. The set of all distinct rotations is denoted by \( R \). We remark that rotation of words has also been used in [7] for finding "minimal starting points" of rotated strings ordered lexicographically, and in [1] for substring matching.

By mending a misspelled textword \( w \) we mean the creation of the set \( S \) of all significant words into which \( w \) can be transformed by assuming \( w \) to have a single error. The set \( S \) is called the set of one-error mendings of \( w \), and each element of \( S \) a one-error mending of \( w \). The RED method mends misspelled words. Of course, selecting the correct word in the context usually depends on linguistic, syntactic, and semantic properties. We do not enter into these considerations here.

When searching for a textword \( w \) using the RED method, the precision is defined as the percentage of the retrieved words that are (1) \( w \) itself or one-error mendings of \( w \), and (2) significant words. Recall is the percentage of the retrieved significant words out of the set consisting of \( w \) and all one-error mendings of \( w \). Different combinations and usage of the RED operators generate different variations of the RED method, which exhibit a tradeoff between space, time, implementability, and precision, though they all have 100 percent recall.

All the variations consist of two parts: (I) a special hash dictionary construction using the RED operators; (II) an appropriate dictionary look-up procedure.

Throughout, \( h \) denotes a hash function.
Variation 1. (I) For every significant word $x = x_1x_2 \ldots x_t$, we store at position $h(D_i(x))$ the triple $(D_i(x), j, p_j)(1 \leq i \leq l)$, where $p_i$ denotes a pointer to the address of $h(D_{i+1}(x))(1 \leq i < l)$ and $p_t$ points to $h(D_t(x))$. The chain $(D_i(x), j, p_j), \ldots, (D_t(x), t, p_t)$ is called the chain of triples of $x$.

Example. For the significant word $ABCDE$, the following triples will be stored in the dictionary: $(BCDE, 1, p_1) (ACDE, 2, p_2), (ABDE, 3, p_3), (ABCE, 4, p_4), (ABC, 5, p_5)$, where the pointers $p_i$ are evaluated during the dictionary construction.

(II) If $w = w_1w_2 \ldots w_t$ is a word that is searched for, then $h(w), h(D_i(w))(1 \leq i \leq l), h(D_1^2(w))(1 \leq i < l)$, and $h(D_1^1(w))$ are looked up, where $D_1^1 = D_1D_2$. By using the index $i$ and the pointer $p_i$, every single error can be found and mended as detailed below.

If $w$ is a word with a missing letter, say the $j$th letter $(1 \leq j \leq |w|)$, then $h(w), h(D_j(w))$ is looked up, where $D_j(w)$ is compared with $h(D_{j+1}(w))$ and $h(D_{j+1}(w))$ is compared with $h(D_j(w))$. The union of the letters of $D_j(w)$ and $D_{j+1}(w)$ (pointed to by $p_j$) enables the reconstruction of a mended word.

If $w$ is a word with a wrong letter, say $w_j(1 \leq j \leq |w|)$, then when $h(D_j(w))$ is looked up, $(D_j(x), j, p_j)$ will be found. The union of the letters of $D_j(x)$ and $D_j+1(x)$ (pointed to by $p_j$) enables the reconstruction of the word $w$.

In this and in the following cases, if $D_j(x)$ is found by $h(D_j(w))$, then for retrieval we also require $i = j$ (or some other match between $i$ and $j$ depending on the error being checked). Otherwise noise is generated instead of correcting a misspelled word. It should be noted that a correctly spelled word which is included in the table will be found at most a single error, then $w$ is a correctly spelled word.

If $w$ is a word with two transposed letters, say $w_j$ and $w_{j+1}$, then $h(D_j(w))$ produces the triple $(D_{j+1}(x), j + 1, p_{j+1})$ and $h(D_{j+1}(w))$ produces the triple $(D_j(x), j, p_j)(1 \leq j < |w|)$, from which $w$ can be reconstructed.

Again every single error can be detected and mended. In this variation, for every significant word $x$, only the forms of RED operators are used, where $[A]$ denotes the contents of address $A$, and $1 \leq i \leq |w|$.

• For a correct word, $R_i(w)$ is compared with $[h(D_iR_i(w))]$.
• For deletion, $R_i(w)$ is compared with $D_i[h(R_i(w))]$.
• For substitution, $D_iR_i(w)$ is compared with $D_i[h(D_iR_i(w))]$.
• For insertion, $D_iR_i(w)$ is compared with $[h(D^2_iR_i(w))]$.
• For transposition, $E_iR_{i-1}(w)$ is compared with $h(R_iR_{i-1}(w))(1 < i \leq |w|)$ and $E_iR_{|w|-1}(w)$ is compared with $[h(R_iR_{|w|-1}(w))]$.

Variation 2. This variation is very similar to Variation 1. In addition to the collection of $|x|$ triples for every word $x$, also $(x, 0, p_0)$ is stored, where $p_0$ is a dummy pointer and the index 0 merely indicates that this triple contains the original word. All pointers of the collection of triples $(D_i(x), i, p_i)(1 \leq i \leq |x|)$ of $x$ point to $(x, 0, p_0)$. The search for $w$ is restricted to $h(w)$ and $h(D_i(w))$ only $(1 \leq i \leq |w|)$, since insertions can be located by $D_i(w)$. Also, a correct word is found immediately by $h(w)$. Here $|x| + 1$ triples need to be stored, but only $|w| + 1$ items have to be located in the hash table.

Variation 3. (I) The hash dictionary contains $R(x)$ and $D_iR_i(x)$ for every significant word $x$ in the form of triples as in Variations 1 and 2.

(II) The search for a word $w$ is performed by using $R(w), D_iR_i(w)$, and $E_iR(w)$. Here again every single error can be detected and mended. The details are similar to those of Variations 1 and 2 and are therefore omitted.

All three RED operators are used in this variation, every significant word $x$ induces storage of $2|x| + 1$ entries in the hash table, and for every word $w$ searched for, $3|w| + 1$ items have to be located in the table. The advantage of this method is that $R$, $D_i$, and $E_i$ are normally easier to implement than $D_i$ and $D_i^2$ for $i > 1$.

Variation 4. (I) For every significant word $x$, the form $R_i(x)$ is stored at address $h(D_iR_i(x))(1 \leq i \leq |x|)$.

(II) For searching a word $w$, the following combinations of RED operators are used, where $[A]$ denotes the contents of address $A$, and $1 \leq i \leq |w|$.

• For a correct word, $R_i(w)$ is compared with $[h(D_iR_i(w))]$.
• For deletion, $R_i(w)$ is compared with $D_i[h(R_i(w))]$.
• For substitution, $D_iR_i(w)$ is compared with $D_i[h(D_iR_i(w))]$.
• For insertion, $D_iR_i(w)$ is compared with $[h(D^2_iR_i(w))]$.
• For transposition, $E_iR_{i-1}(w)$ is compared with $h(R_iR_{i-1}(w))(1 < i \leq |w|)$ and $E_iR_{|w|-1}(w)$ is compared with $[h(R_iR_{|w|-1}(w))]$.

Again every single error can be detected and mended. In this variation, for every significant word $x$, only the forms $R_i(x)(1 \leq i \leq |x|)$ need to be stored in the hash table, rather than triples as in the previous variations. This variation thus requires less space. On the other hand, $4|x| + 1$ items are located and $5|w|$ comparisons are made for every word $w$ searched for.

In all four variations the precision and recall are both 100 percent. Suppose that a smaller precision is acceptable, but the recall is kept at 100 percent, that is, all words that contain a single error of the word searched for are retrieved, but not all retrieved words can necessarily be formed from the word searched for by a single error. Then savings in hash table space or in the number of distinct operators can be achieved. For example, if in Variations 1, 2, or 3 the indexes and pointers are not stored, then some words in addition to the significant words will be retrieved in general. Another way to reduce space requirements for Variations 1 and 3 is to store only two of the forms in two consecutive triples in the chain of triples and to transform the other triples into pairs by...
omitting the forms and pointing to the first of these two triples. For Variation 2, only the form itself need to be stored.

Denote by \( v \) the number of memory accesses required for locating one item in a hash table using a specific hash function. This includes the additional accesses required for resolving hash collisions by methods such as chaining or double hashing. Then the number of memory accesses for retrieving \( w \) is \( t = (2 \mid w \mid + 1) v \) for Variation 1, \((\mid w \mid + 1) v \) for Variation 2, \(3 \mid w \mid v \) for Variation 3, and \( 4 \mid w \mid v \) for Variation 4. The hash dictionary can be stored in an almost full hash table with a good average and worst case \( v \) by using a method such as that proposed by Schmidt and Shamir [6]. Since the same operators are calculated for every word, assembly language routines or even microcoding of them can be prepared, thereby reducing the CPU cost. On the other hand, more “collisions” than in normal hashing can be expected: whenever two distinct dictionary words are transformed into the same string by our operators, both of them are stored, since they are induced by different dictionary words. The problem of locating them is of course taken care of automatically by the collision handling mechanism associated with the hash function, but the number of collisions increases. We have not investigated this effect; instead we wish to thank the referee for pointing out the desirability of doing so.

The RED method can also be extended to detect other types of errors, which are not single errors but occur frequently in optical character recognition, such as changing one character into two other characters (horizontal splitting); changing two characters into one other character (catenation); changing two characters into two other characters (crowding). (The terms are from [5].) This can be achieved, for example, by storing \( D^i(x) \) \((1 \leq i \leq \mid x \mid - 1)\) in the hash table for every significant word \( x \).

Received 4/81; revised 8/81; accepted 3/82

References