Levenshtein distance

The algorithm below simultaneously aligns reference and hypothesis strings and computes the overall word error rate. Partial alignment errors are stored in the matrix R.

Matrix B allows you to backtrack an alignment between strings. An element in B is either "up", "left", or "up-left". When backtracking from $B[n, m]$, at point $B[i, j]$ "up" moves you to $B[i-1, j]$, "left" moves you to $B[i, j-1]$ and "up-left" moves you to $B[i-1, j-1]$. The number of insertion errors equals the number of "left"s on this path, the number of deletion errors equals the number of "up"s, and the substitution errors equals the number of "up-left"s in which the aligned words don't match (i.e., "$REF[i] \neq HYP[j]$").

Input: REF: reference array of words
Input: HYP: hypothesis array of words

begin

    $n \leftarrow$ The number of words in REF
    $m \leftarrow$ The number of words in HYP
    $R \leftarrow zeros(n+1, m+1)$ `// Matrix of distances`
    $B \leftarrow zeros(n+1, m+1)$ `// Backtracking matrix`
    For all i, j s.t. $i = 0$ or $j = 0$, set $R[i, j] \leftarrow \infty$, except $R[0, 0] \leftarrow 0$
    for $i = 1..n$ **do**
        for $j = 1..m$ **do**
            $del \leftarrow R[i-1, j] + 1$
            $sub \leftarrow R[i-1, j-1] + (REF[i] == HYP[j])?0 : 1$
            $ins \leftarrow R[i, j-1] + 1$
            $R[i, j] \leftarrow$ `Min` (del, sub, ins)
            if $R[i, j] == del$ **then**
                $B[i, j] \leftarrow$ ‘up’
            end
            else if $R[i, j] == ins$ **then**
                $B[i, j] \leftarrow$ ‘left’
            end
            else
                $B[i, j] \leftarrow$ ‘up-left’
            end
        end
    end
    Return $100R[n, m]/n$

end

Algorithm 1: Computation of Levenshtein distance, with backtracking.