Dynamic Algorithm of Sequence-Levenshtein distance

Algorithm 1 Sequence-Levenshtein Distance

```
function SequenceLevenshteinDistance($Sequence_1$, $Sequence_2$)
    $l_1 \leftarrow$ length($Sequence_1$)
    $l_2 \leftarrow$ length($Sequence_2$)
    declare $distances : array[l_1 + 1][l_2 + 1]$
    for $i \leftarrow 0, l_1$ do
        $distances[i][0] \leftarrow i$
    end for
    for $j \leftarrow 0, l_2$ do
        $distances[0][j] \leftarrow j$
    end for

    for $i \leftarrow 1, l_1$ do
        for $j \leftarrow 1, l_2$ do
            if $s_1[i-1] = s_2[j-1]$ then
                $cost \leftarrow 0$
            else
                $cost \leftarrow 1$
            end if
            $distances[i][j] \leftarrow$ minimum(
                ▷ Substitution
                $distances[i-1][j-1] + cost$,
                ▷ Insertion
                $distances[i][j-1] + 1$,
                ▷ Deletion
                $distances[i-1][j] + 1$)
        end for
    end for
    $min\_distance \leftarrow distances[l_1][l_2]$

    ▷ Sequence-Levenshtein extension
    ▷ Truncation
    for $i \leftarrow 0, l_1$ do
        $min\_distance \leftarrow$ minimum($min\_distance, distances[i][l_2]$)
    end for
    ▷ Elongation
    for $j \leftarrow 0, l_2$ do
        $min\_distance \leftarrow$ minimum($min\_distance, distances[l_1][j]$)
    end for
    return $min\_distance$
end function
```