Algorithm 13.4 shows how we can do the LZ76 parsing when sources and targets can overlap. It serves to illustrate the key ideas; various improvements can be found in the references we have given.

The algorithm needs the suffix array A of T and its inverse A^{-1}. We also need the structures to find previous and next smaller values on A, psv_A and nsv_A, which require $2n+o(n)$ bits (see Section 13.1.4). The idea is to scan the text left to right, and for each new phrase starting at i, find the suffix array position j pointing to it, $A[j] = i$. Then the positions of A nearest to j (to the left and to the right) with values smaller than $A[j]$ are the suffixes starting in $T[1, i-1]$ that are lexicographically closest to $T[i, n]$, and thus the ones sharing the longest prefix with it. Those positions, *prev* and *next*, are found with $\mathsf{psv}_A(j)$ and $\mathsf{nsv}_A(j)$, respectively. The values $\mathsf{lcp}(T[i,n], T[A[prev], n])$ and $\mathsf{lcp}(T[i,n], T[A[next], n])$ are found directly by scanning T. Then we choose the longest of the two to form the next phrase. If we perform ℓ steps along the scanning process, then the length of the phrase is also ℓ, so the total number of scanning steps is $\mathcal{O}(n)$.

If we use plain representations of A and A^{-1}, then the total space required is $\mathcal{O}(n \log n)$ bits, and the total LZ76 parsing time is $\mathcal{O}(n)$. To reduce space, we can first build a compressed suffix array of T that computes A and A^{-1} in time $t_A = \mathcal{O}(\log_\sigma n)$, for example the one based on bitvectors seen in Section 11.1.2. It can be built in $\mathcal{O}(n)$ time and $\mathcal{O}(n \log \sigma)$ bits (Belazzougui, 2015) (see also Section 11.4). Then we can build the structures psv_A and nsv_A in $\mathcal{O}(n\, t_A)$ time, by accessing each cell of A in time $\mathcal{O}(\log n)$. The parsing itself takes time $\mathcal{O}(z\, t_A + n) = \mathcal{O}(n)$. In total, we perform the LZ76 parsing in $\mathcal{O}(n \log_\sigma n)$ time and $\mathcal{O}(n \log \sigma)$ bits of space.

Algorithm 13.4: Performing the LZ76 parsing of $T[1, n]$ allowing source/target overlaps. We assume that psv_A and nsv_A return 0 and $n + 1$, respectively, when there is no answer.

Input : A text $T[1, n]$.
Output: Outputs the z triples of the LZ76 parsing of T.

Build the suffix array A of T, as well as A^{-1}
Build the structures to compute psv_A and nsv_A
$i \leftarrow 1$
while $i \leq n$ **do**
  $j \leftarrow A^{-1}[i]$
  $prev \leftarrow \mathsf{psv}_A(j)$
  if $prev = 0$ **then** $lp \leftarrow 0$
  else $lp \leftarrow \mathsf{lcp}(T[i, n], T[A[prev], n])$ (computed by brute force)
  $next \leftarrow \mathsf{nsv}_A(j)$
  if $next = n + 1$ **then** $ln \leftarrow 0$
  else $ln \leftarrow \mathsf{lcp}(T[i, n], T[A[next], n])$ (computed by brute force)
  $len \leftarrow \max(lp, ln)$
  if $len = 0$ **then** $pos \leftarrow 0$
  else if $len = lp$ **then** $pos \leftarrow A[prev]$
  else $pos \leftarrow A[next]$
  output $(pos, len, T[i + len])$
  $i \leftarrow i + len + 1$
Free A, A^{-1}, and the structures of psv_A and nsv_A