13.2. REPETITIVE TEXT COLLECTIONS

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 \( A \), \( \text{psv}_A \) and \( \text{nsv}_A \), 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, \( \text{prev} \) and \( \text{next} \), are found with \( \text{psv}_A(j) \) and \( \text{nsv}_A(j) \), respectively. The values \( \text{lcp}(T[i, n], T[A[\text{prev}], n]) \) and \( \text{lcp}(T[i, n], T[A[\text{next}], n]) \) are found directly by scanning \( T \). Then we choose the longest of the two to form the next phrase. If we perform \( \ell \) steps along the scanning process, then the length of the phrase is also \( \ell \), 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 \( \text{psv}_A \) and \( \text{nsv}_A \) in \( \mathcal{O}(nt_A) \) time, by accessing each cell of \( A \) in time \( \mathcal{O}(\log n) \). The parsing itself takes time \( \mathcal{O}(zt_A + n) = \mathcal{O}(n) \). 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 $\text{psv}_A$ and $\text{nsv}_A$ 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 $T$.

1. Build the suffix array $A$ of $T$, as well as $A^{-1}$
2. Build the structures to compute $\text{psv}_A$ and $\text{nsv}_A$
3. $i \leftarrow 1$
4. while $i \leq n$ do
   5. $j \leftarrow A^{-1}[i]$
   6. $\text{prev} \leftarrow \text{psv}_A(j)$
   7. if $\text{prev} = 0$ then $lp \leftarrow 0$
   8. else $lp \leftarrow \text{lcp}(T[i,n], T[A[\text{prev}], n])$ (computed by brute force)
   9. $\text{next} \leftarrow \text{nsv}_A(j)$
   10. if $\text{next} = n + 1$ then $ln \leftarrow 0$
   11. else $ln \leftarrow \text{lcp}(T[i,n], T[A[\text{next}], n])$ (computed by brute force)
   12. $\text{len} \leftarrow \text{max}(lp, ln)$
   13. if $\text{len} = 0$ then $\text{pos} \leftarrow 0$
   14. else if $\text{len} = lp$ then $\text{pos} \leftarrow A[\text{prev}]$
   15. else $\text{pos} \leftarrow A[\text{next}]$
   16. output $(\text{pos}, \text{len}, T[i + \text{len}])$
   17. $i \leftarrow i + \text{len} + 1$
18. Free $A$, $A^{-1}$, and the structures of $\text{psv}_A$ and $\text{nsv}_A$