GENOME-SCALE ALGORITHM DESIGN

by Veli Mäkinen, Djamal Belazzougui, Fabio Cunial and Alexandru I. Tomescu
Cambridge University Press, 2015
www.genome-scale.info

Exercises for Chapter 9. Burrows–Wheeler indexes

9.1 Consider the original definition of the Burrows–Wheeler transform with the cyclic shifts, without a unique endmarker # added to the end of the string. Assume you have a fast algorithm to compute our default transform that assumes the endmarker is added. Show that you can feed that algorithm an input such that you can extract the original cyclic shift transform efficiently from the output.

9.2 Describe an algorithm that builds the succinct suffix array of a string T from the BWT index of T, within the bounds of Theorem 9.6.

9.3 Consider the strategy for answering a set of `occ` locate queries in batch described in Section 9.2.4. Suppose that, once all positions in $\mathsf{SA}_{T\#}$ have been marked, a bitvector $\texttt{marked}[1..n+1]$ is indexed to answer rank queries. Describe an alternative way to implement the batch extraction within the same asymptotic time bound as the one described in Section 9.2.4, but *without resorting to radix-sort*. Which strategy is more space-efficient? Which one is faster? The answer may vary depending on whether `occ` is small or large.

9.4 Recall that at the very end of the construction algorithm described in Section 9.3 we need to convert the BWT of the padded string X into the BWT of string $T\#$. Prove that $\mathsf{BWT}_X = T[n] \cdot \#^{k-1} \cdot W$ and $\mathsf{BWT}_{T\#} = T[n] \cdot W$, where $n = |T|$, $k = |X| - n$, and string W is a permutation of $T[1..n-1]$.

9.5 Assume that you have a machine with P processors that share the same memory. Adapt Algorithm 9.3 to make use of all P processors, and estimate the resulting speedup with respect to Algorithm 9.3.

9.6 Recall that Property 1 is key for implementing Algorithm 9.3 with just one BWT. Prove this property.

9.7 In many applications, the operation $\texttt{enumerateLeft}(i, j)$, where $[i..j] = \mathbb{I}(W, T)$, is immediately followed by operations $\texttt{extendLeft}\,(c, \mathbb{I}(W, T), \mathbb{I}(\underline{W}, \underline{T}))$ applied to every distinct character c returned by $\texttt{enumerateLeft}(i, j)$.

(a) Show how to use operation $\texttt{rangeListExtended}(T, i, j, l, r)$ defined in Exercise 3.12 to efficiently support a new operation called $\texttt{enumerateLeftExtended}(\mathbb{I}(W, T), \mathbb{I}(\underline{W}, \underline{T}))$ that returns the distinct characters that appear in substring $\mathsf{BWT}_{T\#}[i..j]$, and for each such character c returns the pair of intervals computed by $\texttt{extendLeft}\,(c, \mathbb{I}(W, T), \mathbb{I}(\underline{W}, \underline{T}))$. Implement also the symmetric operation `enumerateRightExtended`.

(b) Reimplement Algorithm 9.3 to use the operations `enumerateLeftExtended` and `enumerateRightExtended` instead of `enumerateLeft`, `enumerateRight`, `extendLeft`, and `extendRight`.

9.8 Recall that Property 2 is key for navigating the Burrows–Wheeler index of a labeled tree top-down (Section 9.5.1). Prove this property.

9.9 Consider the Burrows–Wheeler index of a tree $\mathcal{T}$ described in Section 9.5.

(a) Given a character $c \in [1..\sigma]$, let $\overleftarrow{c}$ be the starting position of its interval $\overleftarrow{\mathcal{C}}$ in $\mathsf{BWT}_{\mathcal{T}}$, and let $C'[1..\sigma]$ be an array such that $C'[c] = \overleftarrow{c}$. Describe an $O(n)$-time algorithm to compute C' from C and `last`.

(b) Recall that $\mathbb{I}(v) = [\overleftarrow{v}..\overrightarrow{v}]$ is the interval in $\mathsf{BWT}_{\mathcal{T}}$ that contains all the children of node v, and that $\mathbb{I}'(v)$ is the position of the arc $(v, \texttt{parent}(v))$ in $\mathsf{BWT}_{\mathcal{T}}$. Let $D[1..n+m-1]$ be an array such that $D[i] = \overleftarrow{v}$ where $\mathbb{I}'(v) = i$, or -1 if v is a leaf. Describe an $O(n+m)$-time algorithm to compute D from C', `last` and `labels`. *Hint.* Use the same high-level strategy as the one in Lemmas 8.7 and 9.9.

(c) Give the pseudocode of an $O(n)$-time algorithm that reconstructs the original tree $\mathcal{T}$ from $\mathsf{BWT}_{\mathcal{T}}$ and D, in depth-first order.

9.10 In order to build the Burrows–Wheeler index of a tree $\mathcal{T}$ described in Section 9.5, we can adapt the prefix-doubling approach used in Lemma 8.8 for building suffix arrays. In particular, assume that we want to assign to every node v the order $R(v)$ of its path to the root, among all such paths in $\mathcal{T}$. We define the ith *contraction* of tree $\mathcal{T}$ as the graph $\widetilde{T}^i = (V, E^i)$ such that $(u, v) \in E^i$ iff $v_1, v_2, \ldots, v_{2^{i+1}}$ is a path in $\mathcal{T}$ with $v_0 = u$ and $v_{2^{i+1}} = v$. Clearly $\widetilde{T}^0 = \mathcal{T}$, and E^{i+1} is the set of all pairs (u, v) such that $P = u, w, v$ is a path in $\widetilde{T}^i$. Show how to iteratively refine the value $R(v)$ for every node, starting from the initial approximation $R^0(v) = \overleftarrow{\ell(v)} = C'[\ell(v)]$ computed as in Exercise 9.9. Describe the time and space complexity of your algorithm.

9.11 Assume that we want to generalize the recursive, $O(\sigma+n)$-time algorithm for building the suffix array $\mathsf{SA}_{S\#}$ of a string S to a similar algorithm for building the Burrows–Wheeler index of a labeled tree $\mathcal{T}$ described in Section 9.5. Do the results in Definition 8.9, Lemma 8.10, and Lemma 8.11 still hold? Does the overall generalization still achieve linear time for any $\mathcal{T}$? If not, for which topologies or labelings of $\mathcal{T}$ does the generalization achieve linear time?

9.12 Assume that a numerical value $\texttt{id}(v)$ must be associated with every vertex v of a labeled, strongly distinguishable DAG G: for example, $\texttt{id}(v)$ could be the probability of reaching vertex v from s. Assume that, whenever there is exactly one path P connecting vertex u to vertex v, we have that $\texttt{id}(v) = f(\texttt{id}(u), |P|)$, where f is a known function. Describe a method to augment the Burrows–Wheeler index of G described in Section 9.6 with the values of `id`, and its space and time complexity. *Hint.* Adapt the strategy used for strings in Section 9.2.3.

9.13 Consider the Burrows–Wheeler index of a labeled, strongly distinguishable DAG described in Section 9.6, and assume that `labels` is represented with $\sigma+1$ bitvectors $B_c[1..m]$ for all $c \in [1..\sigma+1]$, where $B_c[i] = 1$ if and only if $\texttt{labels}[i] = c$. Describe a more space-efficient encoding of `labels`. Does it speed up the operations described in Section 9.6?

9.14 Given any directed labeled graph $G = (V, E, \Sigma)$, Algorithm 9.4 builds a reverse-deterministic graph $G' = (V', E', \Sigma)$ in which a vertex $v \in V'$ corresponds to *a subset of* V. Adapt the algorithm so that a vertex $v \in G'$ corresponds to *a subset*

of E. Show an example in which the number of vertices in the reverse-deterministic graph produced by this version of the algorithm is smaller than the number of vertices in the reverse-deterministic graph produced by Algorithm 9.4.

9.15 Recall that in Section 9.6.3 a labeled DAG G is transformed into a reverse-deterministic DAG G', and then G' is itself transformed into a strongly distinguishable DAG G^*. Show that $|G^*| \in O(2^{|G|})$ in the worst case.

9.16 Consider the algorithm to enforce distinguishability described in Section 9.6.3.

(a) Show that this algorithm does not also enforce reverse-determinism.

(b) Assume that the input to this algorithm contains two arcs (u, w) and (v, w) such that $\ell(u) = \ell(v)$: give an upper bound on the number of vertices that result from the projection of u and v to the output.

(c) Modify the algorithm to build a distinguishable, rather than a strongly distinguishable, DAG.

(d) Modify the algorithm to decide whether the input graph is distinguishable or not.

(e) Implement the algorithm using just sorts, scans, and joins of lists of tuples of integers, *without storing or manipulating strings* `path` *explicitly*. Give the pseudocode of your implementation.

(f) Give the time and space complexity of the algorithm as a function of the size of its input and its output.

9.17 Describe a labeled, strongly distinguishable DAG that recognizes an infinite language. Describe an infinite language that cannot be recognized by any labeled, stronglyvdistinguishable DAG.

9.18 Consider the frequency-oblivious representation of a de Bruijn graph described in Section 9.7.1. Show how to implement the function `getArc` when `labels` contains characters in $\Sigma \cup \Sigma'$.

9.19 Recall that marking the starting position of every interval in $\mathsf{BWT}_{T\#}$ that corresponds to a k-mer is a key step for building both the frequency-aware and the frequency-oblivious representation of a de Bruijn graph. Prove that Lemma 9.22 performs this marking correctly.

9.20 Some applications in high-throughput sequencing require the de Bruijn graph of *a set of strings* $\mathcal{R} = \{R^1, R^2, \ldots, R^r\}$. Describe how to adapt the data structures in Sections 9.7.1 and 9.7.2 to represent the de Bruijn graph of string $R = R^1 \# R^2 \# \cdots \# R^r \#$, where $\# \notin [1..\sigma]$. Give an upper bound on the space taken by the frequency-oblivious and by the frequency-aware representation.

9.21 Describe the frequency-oblivious and the frequency-aware representations of the de Bruijn graph $\mathsf{DBG}_{T,k}$ when every string of length k on a reference alphabet Σ occurs at most once in T.