GENOME-SCALE ALGORITHM DESIGN
by Veli Mäkinen, Djamal Belazzougui, Fabio Cunial and Alexandru I. Tomescu
Cambridge University Press, 2nd edition, 2023
www.genome-scale.info

Exercises for Chapter 15. Pangenomics

15.1 Recall the colored de Bruijn graph from Section 15.1.1. If every one of the d colors corresponds to a distinct short read rather than to a whole genome or haplotype, one might consider reducing d by reusing the same color across different reads. Assume that every read corresponds to a path with no repeated vertices, and that colors must allow one to reconstruct all and only the original reads. Which pairs of reads must be assigned different colors?

15.2 Consider the right-to-left de Bruijn graph traversal of Insight 15.1. Describe the details of how to implement it using the suffix tree of the set of walks P. Do you need the whole suffix tree? How would you implement a left-to-right traversal?

15.3 *Compacting* a de Bruijn graph means repeatedly merging into a single vertex every pair of adjacent vertices (u, v) such that v is the only right-neighbor of u, and u is the only left-neighbor of v, until no such merge is possible. This notion is related to the *unitigs* from Section 13.2, and it is especially useful for sets of genomes from the same species. We assume that the vertices of the non-compacted de Bruijn graph are strings of length k.

(i) Describe the compacted de Bruijn graph of the *de Bruijn string* defined in Exercise 11.7, where the graph and the string have the same order k.

(ii) Show how to build the compacted de Bruijn graph directly from the suffix tree of the input sequences in linear time.

(iii) Implement a compacted de Bruijn graph using the frequency-aware representation of the standard de Bruijn graph from Section 9.7.2. How would you enforce the additional constraint that every vertex of the compacted graph must occur in some input sequence?

(iv) Given a substring of length at least k of a vertex of the compacted de Bruijn graph, describe how to find the vertex it belongs to using the previous representation.

15.4 Given a pangenome, it is natural to look for the common parts that are shared by many individuals (the *core genome*), since they most likely code for essential traits. This has obvious applications in finding targets for vaccines or antibacterials. Assume that the pangenome is represented as a colored de Bruijn graph whose nodes correspond to k-mers. Let a *core k-mer* be one with at least c colors, and let a *bridging k-mer* be any k-mer that lies on a walk of length at most ℓ between two core k-mers. Describe an algorithm that computes all core and bridging k-mers in linear time in the number of k-mers. What advantages and disadvantages does this definition of core have?

15.5 Given a reference sequence and a sequence-to-graph aligner, describe an algorithm that grows a pangenome graph by iteratively aligning assembled contigs or long reads to the current graph. Does the graph contain cycles? Does its structure depend on the order with which the sequences are added? How are inversions represented?

15.6 Pangenome graphs are often required to support *positional queries*: Given an integer i and a haplotype T from a large population, find the node of the graph that contains the i-th position of T. This is useful for changing the frame of reference, or for finding the variant closest to i in T. Describe how to support positional queries, assuming that all the haplotypes are represented as sequences of node identifiers in the pangenome DAG, and that the concatenation of all such strings is stored in a BWT as described in Insight 15.4.

15.7 Show how to use the data structures covered earlier in this book to support fast exact pattern matching on the indexable founder graphs studied in Section 15.1.2.

15.8 Fill in the details on how to efficiently construct the indexable founder graphs of Section 15.1.2.

15.9 Given a large set of genomes $\mathcal{S}$ from the same species, assume that we select a small subset $\mathcal{T} \subset \mathcal{S}$ that captures most of the variation, and that we concatenate all the elements of $\mathcal{T}$ into a string T. Assume also that we are given the *relative Lempel–Ziv factorization* of every element of $\mathcal{S} \setminus \mathcal{T}$ with respect to T (see Insight 12.3). Describe a way to build a directed pangenome graph for $\mathcal{S}$ by scanning the factorization and T. Does the choice of the source of a factor affect the graph?

15.10 van Emde Boas tree Consider the pangenome read alignment scheme described in Section 15.2.1. Develop the bookkeeping data structures to map alignment position from the patched sequence B to the position in T.

15.11 Consider the pangenome read alignment scheme described in Section 15.2.1. Show how the wavelet tree can be used to retrieve secondary occurrences, each in $O(\log n)$ time.

15.12 Consider the pangenome read alignment scheme described in Section 15.2.1 and recall Exercise 3.6. Show how to use van Emde Boas trees to retrieve secondary occurrences, each in $O(\log \log n)$ time.

15.13 Show that r-index supports accessing $L[i]$ in $O(\log n)$ time using the data structures we constructed for L', where $L'[1..r]$ is the run-length encoding of $L[1..n]$.

15.14 Show that the data structures of the r-index that support reporting locations can be constructed as fast and as space-efficiently as the structures that support backward search. *Hint:* Mimic the algorithm that builds the samples for the regular succinct suffix array.

15.15 Show that the r-index can be made functionally equivalent to the bidirectional BWT index of Section 9.4. Namely, it suffices to show that operation $\texttt{rangeCount}(L, sp, ep, 0, c-1)$, which counts how many characters there are in $L[sp..ep]$ smaller than c, can be supported in $O(\log n)$ time using only $L'[1..r]$, where L' is the run-length encoding of L. The structure to support the operation should be constructed in $O((\sigma + r) \log n \log \sigma)$ time, and it should take $O((\sigma + r) \log n \log \sigma)$ bits of space.

15.16 Show how to compute values $\kappa(i)$ for prefix pruning for prefixes $P_{1..i}$ of the pattern P in the case of the BWT index on a labeled DAG.

15.17 Give a pseudocode for listing all the subpaths matching a pattern using the BWT index of the labeled DAG. Extend the algorithm given in Section 15.2.3 for listing the matching prefix of a path starting at vertex v with smallest lexicographic order.

15.18 Show that Problem prob:heaviestrecombinationpath can be reduced to the particular shortest-path problem on DAGs from Exercise 4.16.

15.19 Under what assumptions does the shortest-detour algorithm of Section 6.1.2, applied to aligning a haploid prediction to the labeled DAG of a diploid ground-truth, give $O(dn)$ running time, where d is the resulting edit distance and n is the largest of DAG size and haploid length?

15.20 A *series-parallel graph* G is either a single edge that connects a source node to a sink node, or it is the result of merging the sink of a series-parallel graph G_1 to the source of a series-parallel graph G_2 (*series composition*), or of merging the source of G_1 to the source of G_2 and the sink of G_1 to the sink of G_2 (*parallel composition*). A *bubble* is the result of all parallel compositions that affect the same source and sink nodes. Assume that we are given a pangenome graph G that is a series-parallel graph with arbitrary string labels on the nodes, and with a small number of bubbles. Describe a representation of G as a string T, such that all the paths inside the same bubble correspond to a compact block in the BWT of T, and such that one can count the occurrences of a pattern in G via a modified backward search in the BWT of T. How much space does your data structure take? *Hint:* Apply ideas from the *positional BWT* in Section 14.2.2.