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 11. Genome analysis and comparison

11.1 Give an algorithm to construct the indicator bitvector I of Algorithm 11.2 in $O((m+n)\log\sigma)$ time and $O((m+n)\log\sigma)$ bits of space.

11.2 Modify the algorithm for maximal unique matches on two strings to use two bidirectional indexes instead of the indicator bitvector as in our solution for maximal exact matches.

11.3 Recall the algorithm to compute maximal unique matches for multiple strings through document counting in Section 8.4.3. It is possible to implement this algorithm using $O(n\log\sigma)$ bits of space and $O(n\log\sigma\log n)$ time, but this requires some advanced data structures not covered in this book. Assume you have a data structure for solving the *dynamic partial sums* problem, that is, to insert, delete, and query elements of a list of non-negative integers, where the queries ask to sum values up to position i and to find the largest i such that the sum of values up to i is at most a given threshold x. There is a data structure for solving the updates and queries in $O(\log n)$ time using $O(n\log\sigma)$ bits of space, where n is the sum of the stored values. Assume also that you have a succinct representation of LCP values. There is a representation that uses $2n$ bits, such that the extraction of $\mathsf{LCP}[i]$ takes the same time as the extraction of $\mathsf{SA}[i]$. Now, observe that the suffix array and the LCP array, with some auxiliary data structures, are sufficient to simulate the algorithm without any need for explicit suffix trees. Especially, differences between consecutive entries of string and node depths can be stored with the dynamic partial sums data structure. Fill in the details of this space-efficient algorithm.

11.4 Prove Theorem 11.7 assuming that in the output a minimal absent word aWb is encoded as a triplet $(i, j, |W|)$, where i (respectively j) is the starting position of an occurrence of aW in S (respectively Wb in S).

11.5 Prove that only a maximal repeat of a string $S \in [1..\sigma]^+$ can be the infix W of a minimal absent word aWb of S, where a and b are characters in $[1..\sigma]$.

11.6 Show that the number of minimal absent words in a string of length n over an alphabet $[1..\sigma]$ is $O(n\sigma)$. *Hint.* Use the result of the previous exercise.

11.7 A σ-ary de Bruijn sequence of order k is a circular sequence of length σ^k that contains all the possible k-mers over alphabet $\Sigma = [1..\sigma]$. It can be constructed by spelling all the labels in an Eulerian cycle (a cycle that goes through all the edges) of a de Bruijn graph with parameters Σ and k. The sequence can be easily transformed into a string (non-circular sequence) of length $\sigma^k + k - 1$ that contains all the possible k-mers over alphabet Σ.

a) Describe the transformation from circular sequence to string.

b) Show that the number of minimal absent words in this string is σ^{k+1}.

11.8 Assume that you have a set of local alignments between two genomes A and B, with an alignment score associated with each alignment. Model the input as a bipartite graph where overlapping alignments in A and in B form a vertex, respectively, to two sides of the graph. Alignments form weighted edges. Which problem in Chapter 5 suits the purpose of finding anchors for rearrangement algorithms?

11.9 Recall that the formula $p_S(W) = f_F(W)/(|S| - |W| + 1)$ used in Section 11.2 to estimate the empirical probability of substring W of S assumes that W can occur at every position of T. Describe a more accurate expression for $p_S(W)$ that takes into account the *shortest period* of W.

11.10 The *Jaccard distance* between two sets $\mathcal{S}$ and $\mathcal{T}$ is defined as $J(\mathcal{S}, \mathcal{T}) = |\mathcal{S} \cap \mathcal{T}|/|\mathcal{S} \cup \mathcal{T}|$. Adapt the algorithms in Section 11.2.1 to compute $J(\mathcal{S}, \mathcal{T})$, both in the case where $\mathcal{S}$ and $\mathcal{T}$ are the sets of all distinct k-mers that occur in S and in T, respectively, and in the case in which $\mathcal{S}$ and $\mathcal{T}$ are the sets of all distinct *substrings*, of any length, that occur in S and in T, respectively.

11.11 Adapt Lemma 11.11 to compute a variant of the k-mer kernel in which $\mathbf{S}_k[W] = p_S(W)$ and $\mathbf{T}_k[W] = p_T(W)$ for every $W \in [1..\sigma]^k$.

11.12 Adapt Corollary 11.13 to compute a variant of the substring kernel in which $\mathbf{S}_\infty[W] = p_S(W)$ and $\mathbf{T}_\infty[W] = p_T(W)$ for every $W \in [1..\sigma]^k$.

11.13 Given a string S on alphabet $[1..\sigma]$ and a substring W of S, let $\texttt{right}(W)$ be the set of characters that occur in S after W. More formally, $\texttt{right}(W) = \{a \in [1..\sigma] \mid f_S(Wa) > 0\}$. The *kth order empirical entropy* of S is defined as follows:

$$\mathsf{H}(S, k) = \frac{1}{|S|} \sum_{W \in [1..\sigma]^k} \sum_{a \in \texttt{right}(W)} f_S(Wa) \log \left(\frac{f_S(W)}{f_S(Wa)} \right).$$

Intuitively, $\mathsf{H}(S, k)$ measures the amount of uncertainty in predicting the character that follows a context of length k, thus it is a lower bound for the size of a compressed version of S in which every character is assigned a codeword that depends on the preceding context of length k. Adapt the algorithms in Section 11.2.1 to compute $\mathsf{H}(S, k)$ for $k \in [k_1..k_2]$ using the bidirectional BWT index of S, and state the complexity of your solution.

11.14 Let S and T be two strings on alphabet $[1..\sigma]$, and let $\mathbf{S}$ and $\mathbf{T}$ be their composition vectors indexed by all possible k-mers. When the probability distribution of characters in $[1..\sigma]$ is highly nonuniform, the inner product between $\mathbf{S}$ and $\mathbf{T}$ (also known as the D_2 *statistic*) is known to be dominated by noise. To solve this problem, raw counts are typically corrected by *expected counts* that depend on $p(W) = \prod_{i=1}^{|W|} p(W[i])$, the probability of observing string W if characters at different positions in S are independent, identically distributed, and have empirical probability $p(a) = f_{ST}(a)/(|S| + |T|)$ for $a \in [1..\sigma]$. For example, letting $\tilde{\mathbf{S}}[W] = \mathbf{S}[W] - (|S| - k + 1)p(W)$, the following variants of D_2 have been proposed:

$$D_2^s = \sum_{W \in [1..\sigma]^k} \frac{\tilde{\mathbf{S}}[W]\tilde{\mathbf{T}}[W]}{\sqrt{\tilde{\mathbf{S}}[W]^2 + \tilde{\mathbf{T}}[W]^2}},$$

$$D_2^* = \sum_{W \in [1..\sigma]^k} \frac{\tilde{\mathbf{S}}[W]\tilde{\mathbf{T}}[W]}{\sqrt{(|S| - k + 1)(|T| - k + 1) \cdot p(W)}}.$$

Adapt the algorithms described in Section 11.2.1 to compute D_2^s, D_2^*, and similar variants based on $p(W)$, using the bidirectional BWT index of S and T, and state their complexity.

11.15 Show how to implement the computations described in Insight 11.3 using the bidirectional BWT index of $S \in [1..\sigma]^n$. More precisely, do the following.

a) Show how to compute the number of distinct k-mers that appear at least twice (repeating k-mers) in time $O(n \log \sigma)$ and space $O(n \log \sigma)$ bits.

b) Suppose that we want to compute the number of repeating k-mers for all values of k in a given range $[k_1..k_2]$. A naive extension of the algorithm in (a) would take time $O((k_1 - k_2)n \log \sigma)$ and it would use an additional $O((k_1 - k_2) \log n)$ bits of space to store the result. Show how to improve this running time to $O(n \log \sigma)$, using the same space.

c) Show how to compute the Kullback–Leibler divergence described in Insight 11.3, simultaneously for all values of k in $[k_1..k_2]$, in time $O(n \log \sigma)$ and an additional $O(k_1 - k_2)$ computer words of space to store the result.

11.16 Assume that vectors $\mathbf{S}$ and $\mathbf{T}$ have only two dimensions. Show that Equation (11.1) is indeed the cosine of the angle between $\mathbf{S}$ and $\mathbf{T}$.

11.17 Write a program that computes all kernels and complexity measures described in Section 11.2.1, given the suffix tree of the only input string, or the generalized suffix tree of the two input strings.

11.18 Adapt the algorithm in Section 11.2.2 to compute the substring kernel with Markovian corrections in $O(n \log \sigma)$ time and space, where n is the sum of the lengths of the input strings.

11.19 Given strings W and S, and characters a and b on an alphabet $\Sigma = [1..\sigma]$, consider the following expected probability of observing aWb in S, analogous to Equation (11.2):

$$\tilde{p}_{W,S}(a, b) = \frac{p_S(aW) \cdot p_S(Wb)}{p_S(W)}.$$

Moreover, consider the Kullback–Leibler divergence between the observed and expected distribution of $\tilde{p}_{W,S}$ over pairs of characters (a, b):

$$\mathsf{KL}_S(W) = \sum_{(a,b) \in \Sigma \times \Sigma} p_{W,S}(a, b) \cdot \ln \left(\frac{p_{W,S}(a, b)}{\tilde{p}_{W,S}(a, b)} \right),$$

where we set $\ln(p_{W,S}(a, b)/\tilde{p}_{W,S}(a, b)) = 0$ whenever $\tilde{p}_{W,S}(a, b) = 0$. Given strings S and T, let $\mathbf{S}$ and $\mathbf{T}$ be infinite vectors indexed by all strings on alphabet $[1..\sigma]$, such that $\mathbf{S}[W] = \mathsf{KL}_S(W)$ and $\mathbf{T}[W] = \mathsf{KL}_T(W)$. Describe an algorithm to compute the cosine of vectors $\mathbf{S}$ and $\mathbf{T}$ as defined in Equation (11.1).

11.20 Recall that the *shortest unique substring array* $\mathsf{SUS}_S[1..n]$ of a string $S \in [1..\sigma]^n$ is such that $\mathsf{SUS}_S[i]$ is the length of the shortest substring that starts at position i in S and that occurs exactly once in S.

(a) Adapt to SUS_S the left-to-right and right-to-left algorithms described in Section 11.2.3 for computing matching statistics.

(b) Describe how to compute SUS_S by an $O(n)$ bottom-up navigation of the suffix tree of S.

11.21 Show how to compute $\sum_{j=|\ell(u)|+1}^{|W|} \alpha(W[1..j])$ in constant time for all the weighting schemes described in Section 11.2.3. More generally, show that if $\alpha(W)$ depends just on the length of W rather than on its characters, and if $\alpha(W) = 0$ for $|W|$ bigger than some threshold τ, we can access $\sum_{j=|\ell(u)|+1}^{|W|} \alpha(W[1..j])$ in constant time after an $O(\tau)$-time and $O(\tau \log \tau)$-space precomputation.

11.22 Let S and T be strings, and let $D = \{d_1, d_2, \ldots, d_k\}$ be a fixed set of strings with weights $\{w_1, w_2, \ldots, w_k\}$. Show how to compute the kernel $\kappa(S, T)$ in Equation (11.4) restricted to the strings in D, using matching statistics and with the same time complexity as that of Theorem 11.22.

11.23 Given strings S and T and a window size $w \in [1..|S|-1]$, describe an algorithm that computes the kernel $\kappa(S[i..i+w-1], T)$ in Equation (11.4) for all $i \in [1..|S|-w+1]$, performing fewer than m operations per position. *Hint.* Adapt the approach in Theorem 11.22.

11.24 Consider the suffix-link tree SLT_T of a string T, augmented with implicit Weiner links and their destinations. Describe an algorithm that adds to the suffix tree $\mathsf{ST}_{\underline{T}}$ a node for every label W of a node in SLT_T (if string W is not already the label of a node in $\mathsf{ST}_{\underline{T}}$), and that adds to every node of this augmented suffix tree a pointer to its closest ancestor labeled by a string in SLT_T. The relationship between SLT_T and $\mathsf{ST}_{\underline{T}}$ is further explored in Exercise 8.19.

11.25 Given two k-mers V and W on alphabet $[1..\sigma]$, recall that $D_{\mathsf{H}}(V, W)$ is the Hamming distance between V and W, and that $B(W, m)$ is the subset of $[1..\sigma]^k$ that lies at Hamming distance at most m from W. Give a closed-form expression for the size of $B(V, m) \cap B(W, m)$ for every possible value of $D_{\mathsf{H}}(V, W)$ when $m = 2$.

11.26 Given a string S on alphabet $[1..\sigma]$, and given a string W of length k, let $f_S(W, m)$ be the number of substrings of S of length $k + m$ that contain W as a *subsequence*. Consider the following *gapped kernel* between two strings S and T:

$$\kappa(S, T, k, m) = \sum_{W \in [1..\sigma]^k} f_S(W, m) \cdot f_T(W, m).$$

Adapt the approach described in Section 11.2.4 to compute the gapped kernel, and state its complexity.

11.27 In the *neighborhood kernel*, the similarity of two strings S and T is expressed in terms of the similarities of their *neighborhoods* $N(S)$ and $N(T)$, where $N(S)$ contains a given set of strings that the user believes to be similar or related to S. The neighborhood kernel is defined as

$$\kappa_N(S, T) = \sum_{U \in N(S)} \sum_{V \in N(T)} \kappa(U, V),$$

where $\kappa(U, V)$ is a given kernel. Describe how to adapt the approaches described in Section 11.2.4 to compute the neighborhood kernel.

11.28 Prove that $C(ST) + C(U) \leq C(SU) + C(TU)$, by using the symmetry of C.

11.29 Complete the proof of Theorem 11.31 with the two missing cases, using the distributivity of C.

Additional exercises not in the book

11.30 Simulate Algorithm 11.1 (maximal repeat computation using the bidirectional BWT index) on text `TACAGACAC`.

11.31 Read the paper

- Richard Durbin. Efficient haplotype matching and storage using the positional Burrows-Wheeler transform (PBWT). *Bioinformatics*, 30(9): 1266-1272 (2014). `http://dx.doi.org/10.1093/bioinformatics/btu014`.

What is the difference between PBWT and normal BWT? How is the input to the algorithm produced?