IITM-CS6024 : Algorithmic Approaches to Computational Biology Date: Nov 16

Assignment 4 Due Date : Nov 26, 23:55

Roll No:IONdNTPH+ Name:GATTACCA

- Use LaTeX to write-up your solutions, and submit the resulting single pdf file at GradeScope by the due date (Note: **No late submissions** will be allowed, other than one-day late submission with 10% penalty or four-day late submission with 30% penalty! Within GradeScope, indicate the page number where your solution to each question starts, else we won't be able to grade it!).
- Collaboration is encouraged, but all write-ups must be done individually and independently, and mention your collaborator(s) if any. Same rules apply for codes uploaded to HackerRank (i.e., write your own code; we will run plagiarism checks on codes).
- If you have referred a book or any other online material for obtaining a solution, please cite the source. Again don't copy the source *as is* - you may use the source to understand the solution, but write-up the solution in your own words.

---

1. (11 points) [Repetitive BWT]: Let BWT and BWM refer to the Burrows-Wheeler Transform and Matrix respectively. While answering biological questions below, please cite original sources (not Wikipedia!) in proper format.
   (a) (1 point) Reconstruct the string S whose BWT(S) = $nco\$toovican$.
   (b) (2 points) Find a string T distinct from the above string S such that BWM(T) agrees with BWM(S) on its 2^{nd} column (and possibly other columns).
   (c) (2 points) Find a string T' distinct from the above string S such that BWM(T') agrees with BWM(S) on its 2^{nd} and 3^{rd} columns (and possibly other columns).
   (d) (2 points) BWT is used to compress genomes with repeats (i.e., repetitive DNA sequences). What are the different types of repeats found in genomes, and which type is best suited for a BWT-based compression?
   (e) (2 points) We've already seen that repeats cause some issues with *de novo* genome assembly. Please list similar or other distinct issues that reference-based genome assembly using read mapping suffer from due to repetitive DNA sequences.
   (f) (2 points) Which part of a human chromosome contains the most number of repeats, and why? (To answer the "why", you can report evolutionary implications or functional roles/consequences of these repeats.)
2. (10 points) [A fitting code]: A basic understanding of global/local sequence alignment can help you solve other useful variants of sequence alignment like the ones explored in this question.

(a) (7 points) Please mention here briefly how you would solve the fitting alignment problem, and implement it at the contest: https://www.hackerrank.com/cs6024-assignment-4-1. Please read the contest page for the fitting alignment problem definition and instructions for the output format.

(b) (3 points) What changes would you make to your code to solve the overlap alignment problem that is important for *de novo* genome assembly?
An overlap alignment of strings $v = v_1...v_n$ and $w = w_1...w_m$ is a global alignment of a suffix of v with a prefix of w. An optimal overlap alignment of strings v and w maximizes the global alignment score between an i-suffix of v and a j-prefix of w (i.e., between $v_i...v_n$ and $w_1...w_j$) among all i and j.

3. (9 points) [Bla(s)ting thru' sequences]: Having completed a major part of the CS6024 course, you have become a popular sought-after bioinformatician by local health care officials. One morning, you receive a call from an IITM hospital staff that she has collected a viral sample from a patient with grave symptoms and sequence of a gene from that viral sample (which is attached as a sequence in FASTA format in the course moodle). She would like you to use sequence alignment to answer her queries.

(a) (3 points) Which viral species does the sample belong to? Which other viral species is it closest to? Describe briefly how and why you came to your conclusions. Please use the NCBI BLAST toolset.

(b) (3 points) What score and statistical significance fields of the NCBI BLAST output are you using to interpret the results? Provide a concise and clear definition of these score and statistical significance values, and mention the scoring matrix underlying the reported score.

(c) (3 points) You now discover a yet another sequence alignment tool called BLAT. Analyse the viral sample sequence using BLAT and interpret the results. How are the results from this tool for the above questions different? And why?

4. (10 points) [Covering ST and SA]: Let ST and SA refer to suffix tree and suffix array respectively.

(a) (2 points) Briefly describe an algorithm that uses ST to find the longest substring shared by two strings.

(b) (2 points) Briefly describe an algorithm that uses ST to find the shortest substring of one string that does not appear in another string?

(c) (3 points) We saw in class how a simple depth-first traversal of a ST yields the equivalent SA, provided the outgoing edges at every node of the ST are (lexicographically) sorted. Given a ST whose edges at each node are not necessarily sorted, what is the most efficient algorithm you can think of to convert this ST to a lexicographically sorted ST? Specifically, what sorting algorithm would you use at each node and what is its per-node as well as overall running time?

(d) (3 points) The DC3 algorithm for SA construction (recursively) sorts suffixes starting at positions that are **not** multiples of 3 in the first step, and uses the rank of these "sampled" suffixes to sort/merge the remaining suffixes that are at multiples of 3 in later steps. What will happen if you extend this idea to get a DC2 algorithm (which sorts odd-position suffixes in the first step, and uses it to sort/merge even-position suffixes in the next steps)? That is, will the DC2 algorithm's recursion and sort/merge steps be "easier or trickier" to implement than the DC3 algorithm steps? Justify briefly.

(Bonus: What subset of suffixes will you sort in the first step of a DC7 algorithm to get an algorithm similar to the DC3 algorithm? Besides ensuring the recursive step would work, your subset should also satisfy the following key "covering" requirement: For any pair of suffixes S_i, S_j that start at any positions i, j of the string $T = t_1 t_2 \ldots t_m$, there exists a small ℓ s.t. $i + \ell, j + \ell$ are sampled suffix positions. Then, the sort/merge step is efficient since: $S_i \leq S_j \iff (t_i, t_{i+1}, \ldots, t_{i+\ell-1}, rank(S_{i+\ell})) \leq (t_j, t_{j+1}, \ldots, t_{j+\ell-1}, rank(S_{j+\ell}))$.)