Bioinformatics Algorithms Lab
Lab 2

BLAST

Stage 1

For this lab, you will need to load in a "database" of nucleotide sequences in FASTA format. At run time, be sure to pre-process the input database by the method discussed in lecture (finding all high-scoring k-mers). Your code will then read in a query FASTA sequence file and report which sequence(s) in the database had a match to the query, where they match, and generate a consensus for the overlap portions.

Note that this is a multi-step process, starting with the seeding of the search, extension of the search, and finishing with the output of the two sequences and the consensus. Also keep in mind that this lab may take considerable time to debug, as you need to preprocess the database file at every run time.

Requirements:

- Read in the "database" file and parse.
- Read in the query file and parse.
- Correctly identify which sequence(s) in the database locally match all or part of the query sequence.

Stage 2

After implementing your core program, you will need to optimize your code. As you may have noticed, this algorithm is somewhat computationally intensive. To optimize, you need to benchmark key portions of your code. There are many ways to do this. You might use an library/package designed by others to time parts of your code...if you use this approach, please coordinate with other students in class so that only one library/package is used by everyone coding in the same language. You may also use timing functions built into Java/Perl/C to record the stop/start times of key code portions and record the difference...this would be more portable and less likely to create confusion during grading.

Identify which major portions (e.g. subroutines) of your main code are taking the most total time, and report them. There should be at least six major portions recorded. Then, look at the most time intensive portions of your code and look for ways to speed them up. For example, you might find that loading the preprocessing is very time intensive…you might want to think about other ways to implement the resultant data structure that are more time efficient. Feel free to discuss optimization methods with each other in the Discussion group! Just remember that your final code, and any optimization decisions, must be your own and tailored to your own code.

Requirements:

- Printing of key timing information as your code runs. *There must be at least six measurements, together covering all of your code!*

- Submit benchmarking graphs in Excel format to the Dropbox. These should represent timing scores before and after optimization for major portions of your code.

- Report how you improved your code.