CpSc 2120: Algorithms and Data Structures

Instructor: Dr. Brian Dean — Fall 2021
Webpage: `http://www.cs.clemson.edu/~bcdean/` — MWF 9:05-9:55
Handout 2: Lab #1 — Powers 208

1 Detecting the Delta COVID-19 Variant

The goal of this lab is to help everyone get back up to speed with programming in C/C++, and also to practice analyzing the running time of an algorithm. The application of the lab is realistic and timely — the development of effective probes for detecting the infamous Delta variant of COVID-19.

In the directory:

`/group/course/cpsc212/f21/lab01`

you will find the input file `covid.txt`. Make a copy of it in your own directory space.

The input file contains $N = 170$ lines, each with a textual label followed by a genetic sequence, with a space between the two. The genetic sequences are variations[1] of the code for the spike protein for COVID-19 (the protein found on the surface of the virus that helps it attach to cells). Each one is nearly 4000 characters long; they may differ in length slightly due to deletions in the genetic code appearing in some variants. Let M represent the average length of one of these strings.

Each line in the input file is labeled with the variant of COVID-19 it represents, such as "original" (the original strain), "B.1.1.7" (the Alpha variant), or "delta_variant". Eighty samples in the input file represent the Delta variant, and the other ninety samples represent a collection of six other variants.

2 Probe Design

Our goal is to design a short string of $K = 100$ characters that can serve as a "probe" to discriminate Delta variant genetic sequences from non-Delta variant genetic sequences. In an actual biological test, this probe might correspond to a short molecule that can fluoresce when it binds to a genetic sequence. Binding can sometimes occur even if there isn't a perfect match between the probe sequence and the binding target. We will say that our probe sequence "binds" to a target genetic sequence s if the probe matches a substring in s either exactly or with at most one mismatching character. For example, the probe sequence "ACGT" matches "TACACGTAAG" and "CAGGTGTAG", but not "AATCATGGG". The probe might bind at different locations in different sequences from the input.

[1] These were synthetically generated by taking the actual genetic sequences for different COVID-19 variants and applying random mutations (chosen with the same frequencies they appear in actual COVID-19 samples).

To help restrict the search for a probe sequence, we require that it must occur as a substring of one of the input genetic sequences. You should therefore loop over every possible such substring, and pick the best one.

To measure probe effectiveness, consider the two types of errors we might make for any given probe:

- *False positives*, where the probe matches a non-Delta genetic sequence in our input, leading us to erroneously believe it is a sample of the Delta variant.
- *False negatives*, where the probe does not match a Delta genetic sequence, leading us to erroneously believe it is a non-Delta variant.

Based on these, we define the *false positive rate* as

$$FPR = \frac{\text{Number of false positives}}{\text{Number of non-Delta sequences in our input}}.$$

This represents in some sense the probability that we make a mistake when classifying a non-Delta sequence.

Similarly, we define the *false negative rate* as

$$FNR = \frac{\text{Number of false negatives}}{\text{Number of Delta sequences in our input}}.$$

This represents in some sense the probability that we make a mistake when classifying a Delta sequence.

Depending on the situation, it is often the case that false positives and false negatives carry different weights — e.g., it may be worse to erroneously tell someone they have COVID-19 when they do not, rather than erroneously telling them they do not have COVID-19 when they do. In our case, it is not immediately obvious whether false positives are better or worse than false negatives, so we'll make a completely arbitrary assertion that they are twice as bad:

$$\text{Error\_rate} = 2.0 \times FPR + 1.0 \times FNR.$$

Our goal is to find a probe sequence that minimizes this error rate[2].

3 Running Time

As was mentioned, one of the goals of this lab is to practice analyzing the running time of an algorithm. Here, we'd like to write running time in terms of 3 key parameters: N (number of sequence in the input), M (average length of a sequence in the input), and K (length of probe sequence). In terms of these parameters, please determine the worst-case running time of your code, using $O()$ notation.

Note that your code may not behave as its worst-case running time suggests (this happens often, if "worst case" inputs don't occur in practice). Please also determine the "anticipated" running time of your code, also using $O()$ notation.

[2]There are quite a few very important related concepts here that go beyond the scope of this lab and CpSc 2120. For additional enrichment, I'd suggest in your free time reading about "sensitivity versus specificity", "confusion matrices", and "area under the curve". These ideas are used very often in AI to assess the quality of a predictive model (e.g., how well a deep learning model classifies pictures of cats).

4 What to Print for Output

To make your code easy for the graders to grade and for you to debug, please have it print the following things on successive lines of output:

- The best length-K probe sequence it finds.
- The number of false positives and the false positive rate
- The number of false negatives and the false negative rate
- The error rate as computed based on your false positive and negative rates.
- The worst-case running time of your approach.
- The anticipated running time of your approach.

Feel welcome to make your output human-readable, for example by printing out things like "False positives: 7" instead of just "7".

5 Time Constraints

It's likely your code will run quite slowly on the full `covid.txt` data set. For quicker testing, you may want to generate a reduced data set by taking just the first few lines of the file, for example like this:

```
head -n 15 covid.txt > covid-smaller.txt
```

To obtain full credit for this lab, your code should run in less than 30 seconds on a data set consisting of just the first 20 genetic sequences. For fun, based on the running time of your code, you might want to estimate the amount of time it would take to run on the *entire* data set.

6 Coding Considerations

As was mentioned, a primary goal of this lab is to refresh your memory on coding in C++. Here are some guidelines and constraints to keep in mind for writing your code:

- There are no rules about how many comments you need, indentation, etc. This is your code, and you can code in your own preferred style. However, please strive to write clean, simple code for which a third party could easily understand the underlying thought process.
- For now, please do not use any built-in standard template library (STL) data structures like vectors (instead, just use arrays that are dynamically allocated with "new" and "delete"). We'll get to these eventually, but it will be helpful to make sure everyone knows how to use primitive memory allocation first before jumping ahead to STL structures that hide this complexity.
- Please be sure to de-allocate all memory you allocate by the end of your code's execution (we'll always mention in an assignment if this is required).

- Even though the provided input file has 170 lines, please don't make this assumption in your code. Rather, read the input file once to count the number of lines, then allocate sufficient memory to hold the input and read it a second time[3]. To reset the file for reading a second time, you might want to do something like the following:

```
ifstream inputfile("covid.txt");
// Read once to count number of lines
inputfile.clear();
inputfile.seekg(0, ios_base::beg);
// Or alternatively, inputfile.close(); inputfile.open("covid.txt");
// Read second time
// Could be polite and close the file now, but not required
// (it will be closed automatically when your program terminates)
```

- Your code must compile and run on the lab machines (the babbage## machines) to receive credit. Code that doesn't compile will usually receive zero points.

7 Submission and Grading

Submit a single file called `lab1.cpp` containing your solution code. Please don't submit the `covid.txt` file. Code should be submitted electronically on on `handin.cs.clemson.edu`, where we will be turning in all of our work this semester.

Final submissions are due by 11:59pm on the evening of Sunday, August 29. No late submissions will be accepted.

All labs are graded on a scale of zero to ten points, where the point breakdown depends somewhat on the particulars of the lab. For this lab, you will receive 10 points for code that is fully correct and sufficiently fast, and varying amounts of partial credit for incorrect or slow code.

8 Food for Thought

Wouldn't it be nice if we could solve this problem on the full `covid.txt` file quickly? That's a goal to keep in mind as we start learning about various data structures through the first few weeks of class. Other real-world considerations and extensions one could consider for fun: what if a match allowed more than one mismatching character, or even a certain number of deleted characters in one string (so for example "ABCDEF" might match "ABCEF" reasonably well)? What if we are allowed to find a probe with length falling into a certain range (e.g., between 50 and 150 characters) instead of having a specified length? What if we can find 3 probes that collectively can help us guess the right variant classification?[4] – this is possibly realistic in an experimental context if they fluoresce in different colors and can hence be independently measured.

[3] Yes, this would be much less clunky if we were using vectors.

[4] We allowed all of these extensions in an afternoon exercise for top high-school students in USA this past summer who were participating in the USA Computing Olympiad summer training camp. The best solutions they came up with achieved 95% predictive accuracy at identifying the precise variant (not just "Delta" or "not Delta") of a sequence!