CS 170 Algorithms
Fall 2014 David Wagner

# HW9

## Due ~~Nov. 7~~ Nov. 10, 6:00pm

**Instructions.** This homework is due ~~Friday, November 7th~~ Monday, November 10, at 6:00pm electronically. This homework assignment is a programming assignment that is based on a computational biology application. Do this homework individually on your own; you may not work in groups, and you may not discuss your approach with others.

**Background.** Here is some background on DNA sequencing (though strictly speaking you don't need to know it to do this homework). DNA sequencing is the determination of the precise order of the nucleotides in a DNA molecule. A DNA molecule consists of four different bases, namely, adenine (A), guanine (G), cytosine (C), and thymine (T). An example of a DNA strand is CTAGGAGCT, although in reality the number of bases for an actual DNA strand is much larger (on the order of billions). As a result, sequencing the whole DNA is a costly process. Fortunately, researchers have come with alternative methods.

One such method is called shotgun sequencing. First, many copies of the DNA are made (this is possible because copying a DNA molecule is an operation that is much less costly and easier than sequencing). Each copy is then fragmented randomly into pieces. We cannot control how the copies are fragmented; they are randomly split into short contiguous fragments. Next, we sequence each small fragment in parallel; each result is called a "short read." Finally, we use all the short reads to try to reconstruct the original DNA sequence. For this assignment, we will be focusing on this last part, i.e., given all the short reads, we want to re-assemble these short reads to obtain the original DNA sequence. What makes this challenging is that, for each short read, we receive only a short sequence of letters that appears somewhere in the DNA sequence—but we don't know where in the sequence it came from.

Shotgun sequencing was a tremendous advance that helped enable sequencing the entire human genome—and part of what made it possible are algorithms for reconstructing the full DNA sequence, given many short reads. Thus, this is a great example of algorithms in practice. In this homework, you'll solve a slightly simplified version of this DNA assembly problem.

**Problem statement.** We will give you a list of short reads. Your task is to determine the actual DNA sequence. In other words, you are given $n$ strings $s_1, \ldots, s_n$ (the short reads); they are substrings of an unknown string $t$. The goal is to reconstruct $t$ as accurately as possible. We want you to:

1. Design an algorithm for this purpose.
2. Implement your algorithm in any programming language of your choice. Don't worry too much about hand-optimizing the code; it is likely that the choice of algorithm will make a much bigger difference to overall performance than whether you've hand-optimized the inner loop in tight C code.
3. Estimate the running time of your algorithm, assuming you are given $n$ short reads and each short read has length at most $k$.

4. Run and test your algorithm on the data we provide.

To get started, think about how to use what you have learned in class such as divide-and-conquer, graph theory, greedy, dynamic programming to attack the problem.

**Dataset.** You will be given a bunch of data files: `reads1.txt`, `reads2.txt`, ... and `answer1.txt`, `answer2.txt`, ...

Each `readsN.txt` contains a list of short reads from a single DNA sequence. Each line of the file represents one short read. For example, a text file with 264 lines will have 264 short reads. This file is the input to your program.

Each `answerN.txt` contains the full DNA sequence corresponding to `readsN.txt`—or, in other words, the output your program should produce when run on `readsN.txt`. We will provide you the `answerN.txt` file for some of the inputs, so you can test your algorithms and compare your solution to the actual DNA sequence. The output of your program should match the provided `answerN.txt` file exactly, character-for-character, with no extra content.

You should run your program and make sure it produces the correct answer on the test files. Then, run your program on each `readsN.txt` and save its output to `outputN.txt`, and submit those files with your solution.

**Submission.** You must turn in the following files:

1. `writeup.pdf` — a pdf file containing the following.
   (a) An explanation of your algorithm, including the main idea and pseudocode (don't reproduce the full source code; the pseudocode should show the main ideas of your algorithm, but it doesn't need to incorporate every optimization).
   (b) Analysis of the running time of your algorithm. State the asymptotic running time of your algorithm, as a function of $n$ and $k$, and justify your answer.
2. `code/` — directory containing all of your source code.
3. `name.txt` — a text file containing the following:
   (a) Line 1: Your first and last name.
   (b) Line 2: Your login.
4. `output1.txt`, `output2.txt`, ... — each should be text file containing the result of running your program on the corresponding `readsN.txt`. If your program is correct, `outputN.txt` should match `answerN.txt` for each `answerN.txt` file we've provided.

Turn in all of the above files through glookup, by putting all of these files in a directory on your instructional class account and running the `submit` command from within that directory.

*In addition*, submit `writeup.pdf` through Pandagrader (note: you are turning in the `writeup.pdf` through both Pandagrader and glookup).

**Grading.** You will be graded on two things.

1. Accuracy: we will be judging the accuracy of your program's output, i.e., how close it got to the correct DNA sequence. For instance, we might compute the edit distance between the actual DNA sequence and your solution DNA sequence. This also takes into account how many of the inputs your program was able to produce any answer at all, within a reasonable amount of time.

2. Write-up: we will grade your writeup, including the algorithmic ideas you used, the clarity of your explanation, the analysis of its running time, and the efficiency of your algorithm in terms of its asymptotic running time.

**For fun.** The datasets were given to you so that you could achieve very good accuracy. Can you think of cases where your algorithm will do poorly? Can you think of other issues that might come up when trying to sequence DNA? One challenge in practice is that short reads often contain some errors: there is some small error rate in the reads. Can you think of any ways you could extend your algorithms to deal with this challenge? (This is just to think about—no need to answer these questions in your write-up. We aren't cruel—we tried to choose DNA sequences that will avoid the likely bad cases, and for this homework you can assume all reads are perfect and there are no errors—but it's interesting to think about the challenges that arise in practice.)

**Clarifications.** Don't worry too much about corner cases: you can treat the DNA sequence as random and the location of all short reads as random.

For your running time analysis, you might want to analyze your algorithm's expected running time (i.e., average-case running time, assuming a random DNA sequence and random short reads), rather than its worst-case running time.

The data sets are available on the course web page, or in `/home/ff/cs170/hw9/Dataset` on instructional Unix machines.

Revised 11/4 to extend the due date and to add the clarifications section.