Math 2300 **DISCRETE MATH** Fall 2019
Dr. Duval **Project 3**
due Wednesday, December 4

Sequencing by Hybridization

In this project, we'll look at a simple algorithm, using tools we've discussed in class, to assemble a large string of DNA out of smaller fragments.

For our purposes, we'll consider DNA to be strings of letters chosen from the alphabet $\{A, C, G, T\}$, as we did earlier in the semester. An important task in biology, sequencing, is to find the letters of a long sequence of DNA, called the target. One strategy, the "shotgun method", breaks up the target into many smaller overlapping fragments from the target. (Briefly, many copies of the target are made, and then fragmented.) Then we have to do some sort of fragment assembly to deduce the target from the overlapping fragments.

Reassembling the target from these overlapping fragments can be abstracted as the "shortest superstring problem": given a collection of strings [fragments], find the shortest superstring [the unknown target] that contains each of the fragment strings as a substring. This formulation is not exactly what biologists need, for at least two problems. The first is that even small errors in the data (and DNA data is always "noisy", or full of errors) will make the problem either hard or even impossible to solve. The second (and harder to explain) is that it does not do well when there are long repeats in the target. However, we will ignore these problems for now.

Even ignoring the problems of errors and repeats, it can be computationally hard to solve the general shortest superstring problem. We deal with this in a surprising way: we make the fragments even shorter, but of uniform length. The payoff, as we will see, is that we can use our Euler path algorithm (which is very efficient) to solve the problem! The technique is called Sequencing by Hybridization (SBH).

In SBH, we don't look to make several overlapping longer substrings. Instead, we find **all** substrings of some small fixed length. It can be surprising how short this length is, even in real applications. Some modern implementations (with some extra improvements we won't talk about here) look for substrings of only length 5. Part of what makes this work are microarrays, or "DNA chips". It turns out to not be too hard to scan the entire target for the presence or absence of a short string of DNA. Then one may even use parallelization to scan for the presence or absence of all short strings.

Once we have all substrings of length, say b, we make a directed graph as follows. The vertices are the fragments of length $b-1$ (the "-1" is important, don't overlook it); we can get these simply by looking at the first $b-1$, and last $b-1$ letters of each length b fragment we detected. Then we put a directed edge between two vertices if they can be combined to make one of the length b fragments we found. To be more precise, we put a directed edge from vertex u to vertex v for each one of our length b fragments that starts with u and ends with v. (So each fragment contributes one directed edge to the graph.)

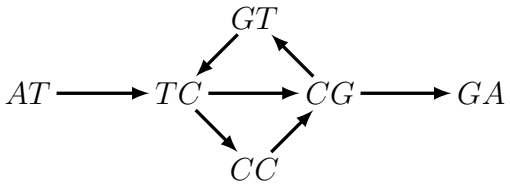
It should be clear that we can trace the entire target string by starting at the vertex corresponding to the first $b-1$ fragment, traveling along the edge corresponding to the first b fragment to the vertex of second $b-1$ fragment, then along the edge of the second b fragment to the vertex of the third $b-1$ fragment, and so on. When we are done, we will

have traveled along all the edges of the graph. That means we are looking for a directed Eulerian path, which we know how to find!

Here is a small example, where $b = 3$. Given the fragments

$$ATC, CCG, CGA, CGT, GTC, TCC, TCG,$$

we construct the graph



It is easy to see that there are two possible Euler paths on this graph, and each corresponds to starting at vertex AT (the only vertex with indegree 0), ending at vertex GA (the only vertex with outdegree 0), and then giving the target string as either

$$ATCCGTCGA \text{ or } ATCGTCCGA.$$

It often happens that SBH does not uniquely identify the target string, but even reducing it down to a few possibilities, as we did here, is an important step.

1. As a warmup, let's start with a target string, and construct its graph. Here $b = 3$, just as in the example. Consider the string

   $$CCTGATTCCATG$$

   (a) List all the substrings of length $b - 1 = 2$. These will be the vertices.

   (b) List all the substrings of length $b = 3$. These will be the directed edges.

   (c) Now draw the corresponding graph. Your first draft of the graph may be messy, but try to draw the graph you submit somewhat neatly, minimizing overlaps.

   (d) Show how an Euler path on this graph corresponds to the original target string.

2. Now we'll try a small example where you have to find the target string! To receive credit for this problem, you must use the Euler path approach described in this assignment.

   Given the $b = 3$-fragments:

   $$AGT, CCG, CGA, CGT, GAG, GCG, GTG, GTT, TGC, TTA$$

   (a) List all the substrings of length $b-1 = 2$ (the length 2 substrings from all of these length 3 substrings). These will be the vertices.

   (b) For each substring of length $b = 3$ listed above, put in the corresponding directed edge. Your first draft of the graph may be messy, but try to draw the graph you submit somewhat neatly, minimizing overlaps.

   (c) Find an Euler path on this graph.

   (d) Use the Euler path to reconstruct the original target string these fragments came from.

3. Now you'll do a much larger example with $b = 4$. You will each receive a different set of fragments of length $b = 4$. Use the techniques in this assignment to reconstruct the target string these fragments came from. If you find several possibilities, just pick one.

   As before, your first draft of the graph will probably be very messy, but try to draw your graph graph somewhat neatly, minimizing overlaps; this may take you several drafts. At a minimum, you **must** draw your graph so that it is easy to see, and follow, your Euler path.

For this assignment, you may write a computer program to help you on any of these problems, but then you have to describe in some detail how you wrote your program. You may **not** use any other software.