Exploring Bioinformatics: A Project-Based Approach

Key for Test Your Understanding Questions, Chapter 8

1. Suppose two sequence reads give GGGGCAGGCC and GCCCCGG. What would be the sequence of the contig produced using the algorithm above?

   There are two possible ways to overlap these fragments:

   ```
   GGGGCAGGCC               GGGGCAGGCC
          GCCCCGG      GCCCCGG
   ```

   But the algorithm discussed here assumes that the larger overlap is the better one, so the first overlap would be used and the contig would be GGGGCAGGCCCCGG.

2. Now suppose you would like your algorithm to account for the possibility that the sequences could come from either strand of the DNA. How would you modify the algorithm to accomplish this? Would the contig resulting from the two sequences in #1 change as a result?

   The algorithm would have to include a function to get the reverse complement of each of the given fragments and then make additional comparisons:

   prefix of 1 with suffix of 2
   ```
        GGGGCAGGCC
   GCCCCGG
   ```

   prefix of 2 with suffix of 1
   ```
   GGGGCAGGCC
          GCCCCGG
   ```

   prefix of 1 with suffix of complement 2
   ```
     GGGGCAGGCC
   CCGGGGC
   ```

   prefix of 2 with suffix of complement 1
   ```
   GGCCTGCCCC
        GCCCCGG
   ```

   prefix of complement 1 with suffix of 2
   ```
        GGCCTGCCCC
   GCCCCGG
   ```

   prefix of complement 2 with suffix of 1
   ```
   GGGGCAGGCC
           CCGGGGC
   ```

   Notice, though, that the last two pairs in this list actually give the same contig, but one is the reverse complement of the other, so really only one from each of these pairs need be considered. The longest of these overlaps would then be chosen. In this case, we get a different contig than in question #1 because considering the other strand allows us to pick a five-nucleotide overlap and get CCGGGGCAGGCC.

3. The above algorithm assumes that the strings cannot be identical and that one cannot be contained completely within the other (one cannot be a substring of the other). But this is a somewhat arbitrary constraint, particularly when comparing a short sequence to a longer contig that has been built. How would you change the algorithm to allow for substrings and identical sequences?

   Both substrings and identical sequences would be found successfully by this algorithm if we simply set the initial *n* (step 2) to the length of the sequence rather than the length of the sequence – 1.

4. Real sequencing data are "noisy"—they can contain incorrect characters due to sequencing errors (for example, the accuracy of most next-generation methods decreases as the fragment length increases) or to ambiguities leading to incorrect base-calling. How would you modify the algorithm so that a perfectly matching overlap is not required, but merely one that exceeds some threshold value? How would incorporating this change affect the number of comparisons that must be made between two sequences?

   In step 4, rather than asking if the number of matching bases = *n*, we could apply the threshold value. Suppose we ask for a 90% match as the minimum acceptable. Then, we'd have a match value = 0.9 × *n* and ask if the count of matching bases is equal to or greater than this number. But, then instead of stopping as soon as a match is made, we should keep going until *n* is less than the number that matched. For example, suppose *n* is 100 and we match 90 out of 100 nucleotides. This is an acceptable match, but it's possible that when *n* goes down to 95, we get a perfect 95/95 match, which is better. So we'd want to keep going until *n* < 90 to see if we can get a better match. (Or maybe even farther: a perfect 75/75 might also be better than 90/100, so some way to make this judgment would need to be built into the algorithm.)