15-110: Principles of Computing

HOMEWORK 07

Due: 10th November, 2020 at 23:59

- You must solve the tasks individually.
- There are 50 points.

1. (20 points) DNA Match

   In the following problems, a DNA sequence is represented by a string composed of the characters
   "A", "C", "T", and "G" only.
   (a) An important problem in working with DNA is subsequence matching. In short, subsequence
   matching determines if a short DNA sequence occurs within a longer sequence. (All the letters
   on the subsequence occur consecutively within the original sequence.) Implement the function
   simpleSubSeqMatch(sequence, subSeq) which, given a DNA sequence sequence and a shorter
   DNA sequence subSeq, returns how many times subSeq occurs within sequence.
   For example:
   - simpleSubSeqMatch("ACCCCTTT", "CCT") should return 1.
   - simpleSubSeqMatch("ACCTACCT", "CCT") should return 2.
   - simpleSubSeqMatch("ACCCCTTTT", "TT") should return 3.
   (b) Sometimes, when doing subsequence matching, we allow partial matches instead of perfect
   matches. A partial match is one where we don’t care about the value of certain characters. If
   we don’t care about a character, we give it the value "N". For example, "ANC" can match any
   three letter sequence that starts with "A" and ends with "C", such as "ATC", "AGC", etc.
   Implement the function subSeqMatch(sequence, subSeq) which, given a DNA sequence sequence
   and a shorter DNA sequence subSeq which may contain N values, returns how many times
   subSeq occurs within sequence.
   For example:
   - subSeqMatch("ACCTCT", "TNT") should return 1.
   - subSeqMatch("ACCTAGCT", "NCT") should return 2.
   - subSeqMatch("ACCCCTTTT", "NTT") should return 3.
   You may find it helpful to create helper functions to solve this task.

2. (30 points) DNA Similarity

   One of the ways scientists use to decide if two species share a common ancestor is by checking how
   similar their DNAs are. In this task you will implement functions for checking DNA similarity.
   The versions you are doing here are simplified versions of what is used in reality.
   The DNA sequences will be represented as strings.
   (a) In this first step, we will determine how many nucleobases (A, C, T or G) are different between
   two DNA sequences. You can assume that they have the same size. Implement the function
   diffNucleobases(seq1, seq2) that returns on how many positions the two sequences have
   different nucleobases.
   For example, diffNucleobases("ACCTAG", "ACCGAG") should return 1, since the sequences only
   differ at the fourth position (index 3).
(b) In reality, though, DNA sequences are not always the same size. In order to find out how similar they are, they need to be aligned somehow. In this case, we will consider alignment to be a partial, in place match between two sequences.

As an example, consider Figure 1. In this figure, we are trying to find the best alignment of "CAT" within "ACCAGT". One potential alignment is shown, which is a two character match starting at index 3. "CAT" matches two of three characters starting at index 3 of the original string. Another potential alignment is shown in Figure 2. In this case, one character matches starting at index 4. The alignment is Figure 1 is better than the alignment in Figure 2, because more characters match.

For simplification, we will only consider sequential alignments that do not “break” the sequence. That means we are only looking for alignments like the ones in Figures 1 and 2 but not alignments that would involve breaking up the string, such as Figure 3.

```
A A C C A G T
C A T
```

**Figure 1: Sample Alignment 1**

```
A A C C A G T
C A T
```

**Figure 2: Sample Alignment 2**

```
A C C A G T
C A T
```

**Figure 3: Sample Invalid Alignment**

Implement the function `simpleAlign(seq1, seq2)` that will return the best sequential alignment between the two sequences. If there are two alignments with the most number of similarities, the function returns the index of the first one (the lower index). For example, the best alignment between $S_1 = ACTGCCCTGTA$ and $S_2 = CAT$ is at index 5 because $CCT$ and $CAT$ differ in only one nucleobasis.

Notice a few things:

- You cannot assume that `seq1` will always be the longer sequence. Your code needs to figure out which one is bigger. (The `min` or `max` functions may help you here.)
- You can use the function from the first part as a helper if you think it will be helpful.