CSCA08 Assignment 1

Due: November 12, 2017. 11:55pm

Introduction

In this assignment, we will be working with strings and lists, combining iteration and selection and dealing with mutability. The goal of this assignment is to ensure that you're comfortable combining all of the elements we have been learning in class, and using multiple tools together in a single function. Once again, your mark will be partially based on building working code, but more than half of your mark will come from documentation and design.

Important Notes

For this assignment, you can use any material covered in class up to and including file i/o. That means you can use loops and selection, as well as any built-in string or list methods that do not require importing. Remember to follow the instructions carefully, and that nowhere in your code should you use `import` or `print` statements.

One thing to keep in mind in this assignment is that genes can be very long. So it will be important that your code be efficient. You don't want to loop over a sequence more times than necessary, and you don't want to create copies of data if it can be avoided.

DNA Sequencing

Now that we have a good understanding of what a gene looks like, we will be working on ways in which genes can be combined and manipulated. Genes are composed of a sequence of nucleotides: adenine (A), guanine (G), cytosine (C) and thymine (T). And once again, we will represent each of these genes by the first letter of their name.

Genes can be paired together by allowing the nucleotides from one gene to pair-bond with the nucleotides from another. Interestingly, guanine will pair with cytosine, and adenine will pair with thymine, but other combinations will not pair. So if we have a gene with the nucleotide sequence `TCAG`[1], it would pair with either `AGTC` or `CTGA` (genes can pair in either direction).

It's also possible for a gene to partially pair with itself, in a process we will call `zipping`. This happens when the nucleotides at either end of a gene form a pair bond, which may in turn allow the next nucleotides in from those genes to bond. This process continues until a pair of nucleotides do not form a bond. For example, the gene `AGTCTCGCT` could form a zip with the first adenine pairing with the last thymine, then the guanine at index 1 bonding with the cytosine at index -2. The next pair inwards would be a thymine and a guanine, which don't pair. So this gene would stop there and only form a zip of length 2.

Scientists are able to splice genes: taking a nucleotide sequence from one gene and replacing it with a nucleotide sequence from another. In order to do this, scientists need to find `anchor sequences` that are the same in both genes, and they can then swap everything in between these anchor sequences[2]. For example, if they wanted to splice the gene codeACATGTGACGT into the gene `TCAGTTACTTGA`, using the anchor sequence `CA` to start the splice and `AC` to end the splice. They would extract `CATGTGAC` from the first gene, and use it to replace the sequence `CAGTTAC` from the second gene, resulting in the new gene `TCATGTGACTTGA`

[1] for this assignment, we will ignore the `AGT` starting codon, and just focus on the rest of the gene

[2] Once again, we're straying from how things actually work in real biology in order to make the problem better fit our needs. This isn't quite how splicing works in the real world, but for the purposes of this assignment, we'll pretend

It's often important to find a specific pattern within a gene. For this purpose, scientists create a `mask`. Masks pair with parts of genes, but do not need to pair with the entire gene. The pairing works in the same manner as normal gene pairing, but with a few interesting additions.

- Scientists can create special nucleotides in masks called `multis` that can mimic the bonding behaviour of multiple nucleotides. For example, we could create a multi that can act like adenine, or like guanine (i.e., it will bond with either thymine or cytosine). We will denote these multis in our gene sequences by listing the nucleotides with which they can mimic in brackets. So for example, a mask consisting of adenine, the mimic just mentioned, and thymine would be written as `A[AG]T`.
- It's also possible to create a nucleotide that will pair with any other nucleotide. We call these special nucleotides `stars`, and we will denote them in our gene sequence with the star character: `*`.
- In order to simplify the encoding of the masks, repeated sequences of nucleotides are denoted by the nucleotide followed by a number, so for example `CCCAGGGGTT` would be represented as `C3AG4T2`.

As an example, the mask: `[AG]C3*` would pair with any sequence starting with either T or C, followed by three G, followed by any other nucleotide.

Your Tasks

For this assignment, you will be required to build the following functions:

- `pair_genes`: Takes in two strings representing genes, and returns True iff the two genes can pair.
- `zip_length`: Takes in a string representing a gene, and returns the maximum number of nucleotide pairs that this gene can zip. For example, if the first 12 nucleotides can zip with the last 12 nucleotides, this function would return 12.
- `splice_gene`: Takes 2 list representations of genes (each element of the list will contain a single character string representing one nucleotide) which we will call `source` and `destination` (in that order) along with a string representing the start and end anchor strings. Splices the subsequence of `source` between the anchor strings into `destination` between the anchor strings. If an anchor occurs more than once in a string, the first shortest sequence should be used. If the anchors do not appear in order in both strings, no changes should be made to the genes.
- `match_mask`: Takes in a string representation of a gene and a mask and returns the index of the first nucleotide in the sequence that is matched by the mask. (If the mask matches multiple sequences, return the one with the lowest index) or -1 if the mask does not match anywhere in the gene.
- `process_gene_file`: Takes in a file handle for a file containing one gene per line, a string representing a gene and a string representing a mask. Returns a tuple `(p, m, z)` where `p` = the first gene that can pair with the input gene string, `m` = the first gene that matches the mask, and `z` = the longest gene zip found up in any gene up to and including the point where both `p` and `m` were found. If no genes match the given gene or mask, `-1` is returned in place of `p` or `m`.

Marking

What to Submit

Submit `a1.py` and `a1.test` on MarkUs. Your files must be named exactly as given here (check that MarkUs says you have submitted all required files after you're done submitting).