Ashkan Bigdeli

Cscie10b

Term Project

Sequence Analyzer

For my term project I set out to make a GUI that will analyze and annotate genetic sequences. The program will identify common features, create the complementary DNA sequence, retrieve their location within a genetic sequence and set the text for easy reading by the user.

Specifications:

1. The GUI is straight forward. Simply take any text and paste it into the window. The program will parse out any genomic sequence and provide you with several key features.
   1. First you will be provided with the original sequence you input with space or needless characters trimmed. Not you may see some non-base characters at the end, but these will be removed when you are given your complimentary strand.
   2. Next it will convert this sequence to its complimentary strand. A’s match T’s and G’s match to C’s in genomic data. So if you paste in AGT, the output will be TCA.
   3. After displaying the original and complimentary strands the program will find features for you. In informatics finding features for annotation is critical to making quick work of molecular bio. In this program you will find a short list of common primers (sites to read DNA), common cut sites (areas where you may manipulate DNA) and start and stop codons (areas where gene expression may begin or end). All of these features will be output beneath your complimentary strand and their location will be displayed with it.
      1. EXAMPLE: PRIMER M13R: Base 17 to 35
      2. This is important as it lets you know the expected size of PCR products and digestions.

List of common priming sites given to you by SequenceAnalyzer v 1.0 :

1. M13R – CAGGAAACAGCTATGAC
2. CMV –F – CGCAAATGGGCGGTAGGCGTG
3. SmaI – GACTATCATATGCTTACCGT
4. EcoRV – TAATACGACTCACTATAGGG
5. SP6 – ATTTAGGTGACACTATAG

For a more detailed look at primers please see : <http://www.contexo.info/DNA_Basics/Primers.htm>

List of common DNA Cut sites :

1. EcoRI – GAATTC
2. NotI – GCGGCCGC
3. BAMHI – GGATCC
4. HindIII – AAGCTT
5. TaqI – TCGA

For a more detailed look at Enzyme cut sites and what they are please see : <http://en.wikipedia.org/wiki/Restriction_enzyme>

List of codons used within :

1. Start – ATG
2. Stop – TAG

Codons are not as easily explained through a wiki or detail page, which is why I only picked start and stop: <http://en.wikipedia.org/wiki/DNA_codon_table>

Test my code:

Please separate any full length sequences using the > character. Here are two sequences containing 1 sequence, 1 start, 1 stop and 1 cut site. Feel free to come up with as many character or non-character variations to test the code.

CTTCAAGGAGAATC ATG ACGCTTCTTTCTAAAGATGGATTCACCATTTAAAACAGAGCTCTGG

GAGCCTTTCGGCAAATCTTGAAAGCTGCACGGTGCAGAGACATGGATGTGACTTCCCAAG

CCCGGGGCGTGGGCCTGGAGATGTACCCAGGCACCGCGCAGCCTGCGGCCCCCAACACCA

CCTCCCCCGAGCT TAATACGACTCACTATAGGG TGTCCCACCCGCTCCTGGGCACCGCCCTGGCCAATGGGACAG

GTGAGCTCTCGGAGCACCAGCAGTACGTGATCGGCCTGTTCCTCTCGTGCCTCTACACCA

TCTTCCTCTTCCC TCGA CATCGGCTTTGTGGGCAACATCCTGATCCTGGTGGTGAACATCAGCT

TCCGCGAGAAGATGACCATCCCCGACCTGTACTTCATCAACCTGGCGGTGGCGGACCTCA

>

CGCTGGAGATGCAAGGTGCTGGTGGGTCTGAGCTGGACGTCGCGGTGTGTCCTCTGTGCC

CACGGTCTGAGCTAGCTAGCGCACCGCCGAGTTA CGCAAATGGGCGGTAGGCGTG AAGAGGAGAAGGAAAACATGCTGCTC

TGGTG GGATCC CACGCCTGAGCGTCCTCCATCTTCCAGGATGGCAGCAATGGCGCTGTGCGGCCTC

ACCAGGCCCACGAGGAGCAGCAGCGCTCGGCCCGGAGCAGCAGGAAGGCCCCTCTGTGGA

GCGCCCGCCGTCTGCTC TAG CGGGGTGGTTCAGTCACTGCTTGTTGACATCAACATGGCAATT

GCACTCATGTGGACTGGGACCGTGCGAGCTGCCGTGTGGGTTAGTCGGGTGCCAGGACAA

TGAAATACTCCAGCACCTGTGGCTGACGAATTTGTTTCTACAGAAATAACAGCTGGG

Detailed method explanations for important methods:

1. parseSeq: I view this as the crown jewel of my program. This method can be used to parse out genomic sequences from in theory everything. It is actually not all that necessary within this program as I could have put the responsibility on the user to input the correct formats. However, generally speaking there are programs that do what SequenceAnalyzer v 1.0 does very well. This method will allow someone to input garbage and walk away with sequences. This is especially useful as this method can be re-purposed in my work environment to call articles, websites or the like for genetic data.
2. isBase: This is another useful method, it simply returns true if you feed it a character and that character can be considered a base. To keep genomic integrity intact, if we are uncertain of a base we can simply insert an N to maintain our correct positions.
3. getCompBase: This is another method I plan to use later, it works behind the scenes in many informatics programs I assume as you must know the complimentary strand to perform simple tasks. Typically you do not display it to the user as it can be easy to derive mentally when needed, but it is necessary to have such a method to perform alignments and other common informatics methods.
4. findFeature: This method is specific to this program as it takes a clunky 2d array to provide input to the user correctly. I am however better for writing it. Working through the 2d array to match derive a common primer name (something far more useful than just a sequence) to the user was a challenge. I should say I am very happy about how this one method is called 3 times to find features. Just a few months ago I likely would have written 3-4 separate methods to achieve this.

Challenges and short comings and final thoughts:

This program taught me a world about design. You’ll notice my program has a major bug, if your sequence has multiple priming sites it will only give you the first it finds, this is because I stayed married to one approach early on assuming it would fit effortlessly with the next. I ended up painting myself in a corner with how I opted to display my user output. This was a lot of little features to implement and display in one area. If I could go back I would have displayed one sequence at one time. This is how most of these programs work and I thought I could bind the functionality of my parseSeq into the rest of my program. The goal of my program was not to parse sequences despite how useful this function is. I should have simply taken one sequence and given you the results writing it to a file or setting it to the text for you to retrieve. However I opted to take in multiple sequences and in the end it end up creating a bug.

Please visit this site and paste in the sequences I have given you above.

<https://www.addgene.org/analyze-sequence/#>

You’ll notice it takes only 1 input.

Or perhaps visit this one:

<http://blast.ncbi.nlm.nih.gov/Blast.cgi?PROGRAM=blastn&PAGE_TYPE=BlastSearch&LINK_LOC=blasthome>

You’ll notice it does not provide you with features.

What has this taught me? Do not try and make one tool to do it all. Make several tools and call on them as needed. It is something that has been preached throughout this course, and it was great to learn the hard way first hand. I had to learn to fail fast when trying to work through so many arrays together. In retrospect this is certainly not the best approach and had I kept it simple it would not have been the one I took. Array lengths can vary when you add to them as needed and when I tested my code with varying inputs I broke it over and over again. I have fixed the bugs, but creating a simpler tool to start, like reading file input, would have prevented needing to do so. I should have provided a better frame work for 1 input and then worked to do that many times rather than building an input that could take a bulk, and then have to handle that.

There are many things I would change about the project if I could do it again, but I learned a lot and am proud of the end product. If I had to do it twice it would take a quarter of the time and be twice and refined, but handling the complexity of programming and coupling it with the complexity of science is why I took this course.