ESC190 Lab 5

Bit Manipulation & Potatoes of Mass Destruction

Due: April 9th, 2020 at 23:59

Backstory

The year is still 2060. The stings of defeat from the events of "ESC190 Lab 2: The Last Library" and "ESC190 Lab 4: Hash Tables" burn like obscenely large papercuts on the thumbs of the Government. Instead of gracefully accepting failure, the Government has decided to weaken the Resistance by releasing a new virus, dubbed the "Spud Lite Virus" (SLV), into a subsection of the next crop of potatoes. Fortunately, the Resistance informant has caught on to this heinous ploy and was able to retrieve a sample of SLV from the Government laboratories for analysis. Rather than deprive themselves of delicious potatoes, the Resistance has tasked you with developing computational tools to aid the Resistance Microbiologists in developing an SLV vaccine before the Government's potatoes of doom are ready to be harvested.

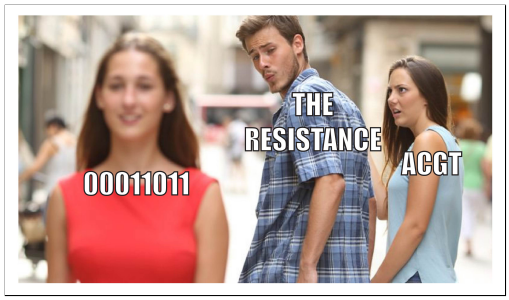
Provided Files

Run `git pull` to load the following files:

- `fullSLV.txt`, `partialSLV.txt` \\These files contain full and partial SLV nucleobase sequences. You may find it helpful to initially test your functions on the shorter sequence.
- `codons.txt` \\This file contains all the binary codon translations.
- `lab5.c` \\Implement the specified functions in this file for grading. You may want to also write `main.c` and `lab5.h` files for testing purposes (these files will not be graded).

Tasks

Part 1. Genetic Encryption



DNA (deoxyribonucleic acid) molecules are double-stranded helical structures that function as genetic instructions for biological cells. SLV injects its DNA into unsuspecting cells, which read the DNA and unknowingly follow its instructions to assemble more SLV for infecting other cells. DNA strands are composed of pairings of nucleobases ("base pairs"), but can be represented more simply as a single-stranded sequence of 4 nucleobases: adenine (`A`), cytosine (`C`), guanine (`G`), and thymine (`T`). As a security measure, the Resistance requires all DNA sequences to be converted from nucleobase sequences to binary numbers as per **Table 1**. For example, the nucleobase sequence `GCCACCATG` should be converted to the binary sequence `100101000101001110`. Therefore, you will need to implement the following functions to seamlessly switch between the two sequence formats:

Table 1. Nucleobase to binary conversion.

Nucleobase	Binary
A	00
C	01
G	10
T	11

`void encodeNuc(char *filename)`
This function reads the nucleobase sequence from a text file and generates a new text file containing the corresponding binary sequence.

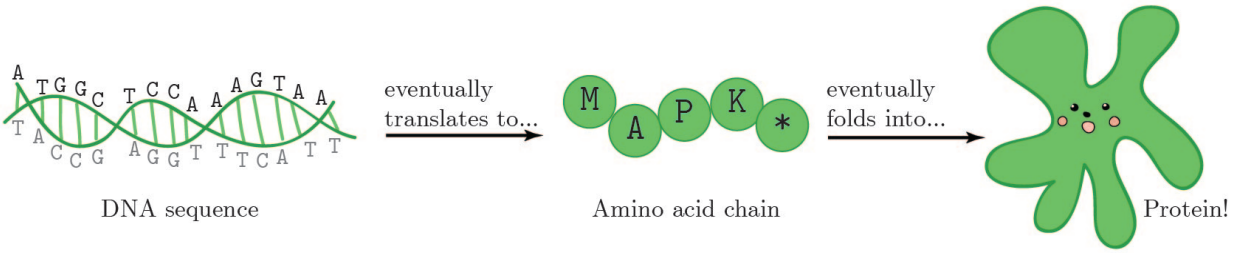
- **Input**: `filename`, the filename of the text file containing a nucleobase sequence to be read and encoded as binary.
- **Output**: Write the binary sequence to a text file named `bfilename`, where `filename` is the input filename. For example, calling `encodeNuc("thisisthefilename.txt")` should generate an output text file named `"bthisisthefilename.txt"`.

`void decodeBin(char *filename)`
This function reads the binary sequence from a text file and generates a new text file containing the corresponding nucleobase sequence.

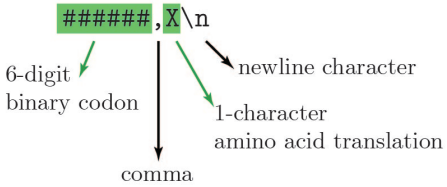
- **Input**: `filename`, the filename of the text file containing a binary sequence to be read and decoded to a nucleobase sequence.
- **Output**: Write the nucleobase sequence to a text file named `nfilename`, where `filename` is the input filename.

Part 2. Codon Translation

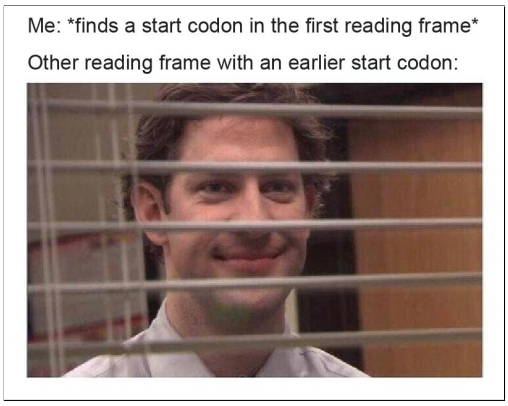
Cells interpret DNA sequences in triplets of bases, known as codons, and assemble the corresponding amino acid translations of the codons into chains that fold to become functional proteins. Just as nucleobases are the units of DNA, amino acids are the units of proteins!



For your reference, `codons.txt` contains binary codon translations in the form:



These translations are based on the nucleobase codon translation table shown in **Table 2** of the **Appendix**.



Reading nucleobases in triplets means that there are three possible interpretations of any DNA sequence, i.e. there are three "reading frames" for DNA, and three corresponding amino acid sequences that can be translated. Only one reading frame may be active for translation at any position in the DNA sequence. The active reading frame is determined by the "start" codon (codon: `ATG` = amino acid `M`) and "stop" codons (codons: {`TAA`, `TAG`, or `TGA`} = amino acid `*`). These codons also indicate the beginning and ending of any proteins to be made. The first start codon to appear in the sequence sets the active reading frame and is the first amino acid of the protein. The first in-frame stop codon to appear after a start codon is the last amino acid of the protein. The next start codon to appear after a stop codon may reset the active reading frame for the next protein.

In the DNA sequence shown below, the first start codon among the three reading frames is at the 4^{th} nucleobase. This establishes reading frame 1 as the active frame for translating the first protein. The last amino acid of this protein is the first stop codon to appear in reading frame 1 after the start codon, found ending at the 18^{th} nucleobase. This first protein, highlighted below, will be 5 amino acids long (`MAPK*`). The next start codon to appear in the DNA sequence after the first protein is at the 21^{st} nucleobase, so reading frame 2 will be the next active frame for translating the next protein, also highlighted below.

```
DNA Sequence:          ACCATGGCTCCAAAGTAACCATGGATGC...

Reading Frame 1:       ACC ATG GCT CCA AAG TAA CCA TGG ATG C...
Amino Acid Sequence 1: T   M   A   P   K   *   P   W   M

Reading Frame 2:       AC CAT GGC TCC AAA GTA ACC ATG GAT GC...
Amino Acid Sequence 2:    H   G   S   K   V   T   M   D

Reading Frame 3:       A CCA TGG CTC CAA AGT AAC CAT GGA TGC...
Amino Acid Sequence 3:   P   W   L   Q   S   N   H   G   C
```

Proteins are the workforce of the cell, responsible for carrying out nearly all cellular functions. Identifying which proteins are present in the SLV DNA sequence will be critical to understanding how SLV operates in cells. The Resistance needs you to locate and isolate the proteins in a given DNA sequence by implementing:

`void findProtein(char *filename, int checkPos, int proteinInfo[])`
This function reads a binary sequence from a text file and identifies the first protein found at or after a given nucleobase position.

- **Inputs**:
  - `filename`, the filename of the text file containing a binary sequence to be read.
  - `checkPos`, the nucleobase position in the sequence to start checking for a protein.
  - `proteinInfo`, a 2-element array to be updated with protein information.
- **Output**:
  - For the first protein to appear at or after the `checkPos` position, set:
    - `proteinInfo[0]` = the nucleobase position of the start of the start codon.
    - `proteinInfo[1]` = the amino acid length of the protein.
    - `proteinInfo[]` = {`0,0`} if there is no protein.

`void proteinReport(char *filename)`

This function reads a binary sequence from a text file and generates a new text file containing information about all the protein regions in the sequence.

- **Input**: `filename`, the filename of the text file containing a binary sequence to be read.
- **Output**: Write the protein location information for all proteins in the sequence into a text file named `rfilename`, where `filename` is the input filename. Record each protein region into the text file in the format `a,b\n` where `a` is nucleobase position of the start of the start codon, `b` is the amino acid length, separated by a comma, and `\n` is a newline character. If no protein regions exist, write `0,0\n` into the text file.

`void isolateProtein(char *filename, int proteinInfo[])`

This function reads a binary sequence from a text file and generates a new text file containing the amino acid translations of a specified protein.

- **Inputs**:
  - `filename`, the filename of the text file containing a binary sequence to be read.
  - `proteinInfo`, as before, a 2-element array containing protein information.
- **Output**: Write the amino acid translations for the protein specified by `proteinInfo` into a text file named `pfilename`, where `filename` is the input filename. For example, if `filename = "hi.txt"`, then `pfilename` should be `"phi.txt"`. `pfilename` should contain a sequence of characters starting with `M` and ending with `*`, i.e. no spaces or newline characters. `pfilename` should be an empty text file if `proteinInfo[]` = {`0,0`}.

Part 3. Point Mutations

Vaccines work by training your immune system to recognize a new disease and build defensive response mechanisms. Before a vaccine can be distributed, it must be rigorously tested to ensure that it will work without causing disease or other unsafe side effects. With access to the SLV DNA sequence and the tools to identify the proteins that will be generated from SLV DNA, the Resistance is poised to explore deactivated SLV proteins as potential vaccine candidates. The ideal deactivated SLV protein will have lost all SLV-inducing functionality while still maintaining enough resemblance to the original protein to adequately train the immune system to recognize and neutralize the original protein. The Resistance Microbiologists have identified several promising single nucleobase mutations, called point mutations, to deactivate SLV proteins. There are three types of point mutations we are interested in applying:

- **Deletion**, where a single nucleobase is removed from the sequence. This results in a "frame shift", where the reading frames appear to be swapped after the mutation site.
- **Insertion**, where a single nucleobase is inserted into the sequence. This also results in a frame shift.
- **Substitution**, where a single nucleobase in the sequence is replaced with a different nucleobase. This does not cause a frame shift, but can potentially alter the amino acid translation at the mutation site.

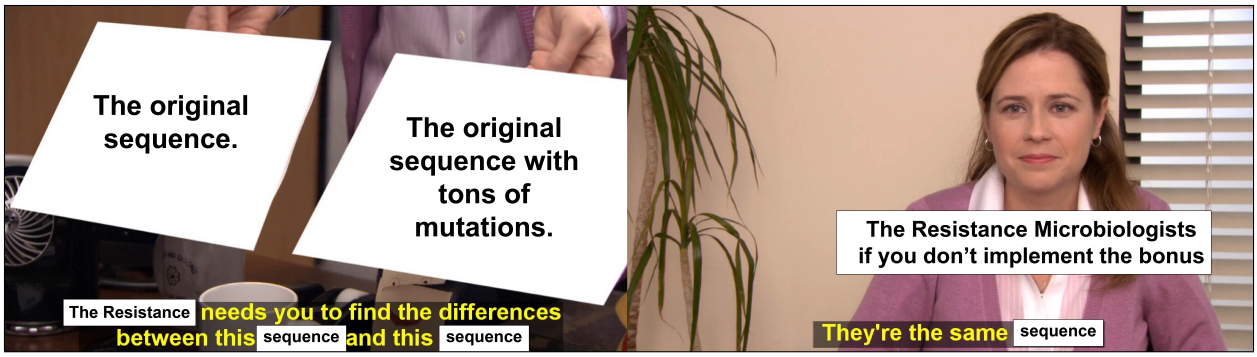
The Resistance needs you to implement the following to annotate DNA sequences:

`int genMutant(char *filename, int mutation[])`
This function reads a binary sequence from a text file and generates a new text file containing a copy of the sequence with a specified point mutation.

- **Inputs**:
  - `filename`, the filename of the text file containing a binary sequence to be read.
  - `mutation`, the point mutation to apply, given as a 3-element array where:
    - `mutation[0]` = the nucleobase position to mutate.
    - `mutation[1]` = the type of point mutation: `0`=deletion, `1`=insertion, `2`=substitution.
    - `mutation[2]` = the original nucleobase to be deleted, or the new nucleobase to be inserted or substituted: (`0`=`A`, `1`=`C`, `2`=`G`, `3`=`T`).
- **Outputs**:
  - Return `0` to indicate successful point mutation annotation and write the mutated binary sequence into a text file named `mfilename`, where `filename` is the input filename.
  - If there are any issues applying the mutation, return `1` and write the original binary sequence into a text file named `mfilename`, where `filename` is the input filename.

Bonus

This bonus portion of the lab is completely optional, but if you successfully implement this bonus you will receive an extra 1% added to your overall grade!



The ability to align two sequences and detect any differences between them will be necessary in order to confirm the presence of any applied point mutations as well as track any spontaneous mutations over time. Therefore, the Resistance would like you to implement:

`int checkMutant(char *oriFilename, char *mutFilename)`
This function reads the binary sequences from two text files and generates a new text file containing information about any point mutations detected. Only the first protein detected in each of the two sequences need to be compared, which may be located at different nucleobase positions in each sequence.

- **Inputs**:
  - `oriFilename`, the filename of the text file containing the original binary sequence to be read.
  - `mutFilename`, the filename of the text file containing a potentially mutated version of the `oriFilename` sequence. You can assume that the first start codon will not be mutated and there will not be more than 1 point mutation per codon from the `oriFilename` sequence.
- **Output**:
  In a text file named `cmutFilename`, where `mutFilename` is the input filename, write the point mutations detected in the format `a,b,c\n` where:

- `a` = the nucleobase position of the mutation (let the start of the start codon be position 1).
- `b` = the type of point mutation: `0`=deletion, `1`=insertion, `2`=substitution.
- `c` = the original nucleobase to be deleted, or the new nucleobase to be inserted or substituted: (`0`=`A`, `1`=`C`, `2`=`G`, `3`=`T`).
- `a`, `b`, and `c` are comma separated and `\n` is a newline character.
- If there are no mutations, `cmutFilename` should be an empty file.

Return the edit distance between the nucleobase sequences of the first proteins derived from `oriFilename` and `mutFilename`. The edit distance is the least number of nucleobase changes needed to make the sequences the same.

Appendix

Further Reading

While the backstory for this lab is a work of fiction, the biology concepts introduced {viruses, cells, DNA, amino acids, proteins, etc.} are a simplified version of the real science. A major component not discussed here is the intermediate transcription of DNA to RNA prior to translation, which you will have the joy of learning more about in BME205!

Codon Translation Table

Table 2. Nucleobase codon to amino acid translation.
Recall that codons consist of three nucleobases and translate to a single amino acid. To translate a codon, locate the first nucleobase on the left side of the table, move across horizontally to the column corresponding with the second nucleobase, and finally shift down vertically as per the third nucleobase to identify the corresponding amino acid translation.

1st Base	2nd Base								3rd Base
	T		C		A		G		
T	TTT	Phenylalanine (F)	TCT	Serine (S)	TAT	Tyrosine (Y)	TGT	Cysteine (C)	T
	TTC		TCC		TAC		TGC		C
	TTA	Leucine (L)	TCA		TAA	Stop (*)	TGA	Stop (*)	A
	TTG		TCG		TAG		TGG	Tryptophan (W)	G
C	CTT	Leucine (L)	CCT	Proline (P)	CAT	Histidine (H)	CGT	Arginine (R)	T
	CTC		CCC		CAC		CGC		C
	CTA		CCA		CAA	Glutamine (Q)	CGA		A
	CTG		CCG		CAG		CGG		G
A	ATT	Isoleucine (I)	ACT	Threonine (T)	AAT	Asparagine (N)	AGT	Serine (S)	T
	ATC		ACC		AAC		AGC		C
	ATA		ACA		AAA	Lysine (K)	AGA	Arginine (R)	A
	ATG	Methionine/Start (M)	ACG		AAG		AGG		G
G	GTT	Valine (V)	GCT	Alanine (A)	GAT	Aspartic acid (D)	GGT	Glycine (G)	T
	GTC		GCC		GAC		GGC		C
	GTA		GCA		GAA	Glutamic acid (E)	GGA		A
	GTG		GCG		GAG		GGG		G

This image is based on the table available at: https://en.wikipedia.org/wiki/DNA_codon_table

Meme Image Sources

- Evil Kermit [Digital Image]. (2014). Retrieved from https://bit.ly/2UgHruQ
- Distracted Boyfriend [Digital Image]. (2015). Retrieved from https://bit.ly/3bpaVwl
- Jim Halpert Smiling Through Blinds [Digital Image]. (2009). Retrieved from https://bit.ly/3ahWQRe
- Tom Hiddleston at the US Open Tennis Championship [Digital Image]. (2019). Retrieved from http://dailym.ai/2Ug6qhR
- They're The Same Picture [Digital Image]. (2018). Retrieved from https://bit.ly/39lov2g