6.10 PROJECTS

Project 6.1 Polarized politics

The legislative branch of the United States government, with its two-party system, goes through phases in which the two parties work together productively to pass laws, and other phases in which partisan lawmaking is closer to the norm. In this project, we will use data available on the website of the Clerk of the U.S. House of Representatives (`https://clerk.house.gov/Votes`) to analyze the voting behavior and polarization of the U.S. House over time.

Part 1: Party line votes

We will call a vote *party line* when at least half of the voting members of one party vote one way and at least half of the voting members of the other party vote the other way. In the House, when the vote of every member is recorded, it is called a *roll call* vote. Individual roll call votes are available online at URLs that look like the following:

`https://clerk.house.gov/cgi-bin/vote.asp?year=<year>&rollnumber=<number>`

The placeholder `<year>` is replaced with the year of the vote and the placeholder `<number>` is replaced with the roll call number. For example, the results of roll call vote 194 from 2010 (the final vote on the Affordable Care Act) are available at

`https://clerk.house.gov/cgi-bin/vote.asp?year=2010&rollnumber=194`

Run the following program to view one of these files:

```
import urllib.request as web

def main():
    url = 'https://clerk.house.gov/cgi-bin/vote.asp?year=2010&rollnumber=194'
    webpage = web.urlopen(url)
    for line in webpage:
        line = line.decode('utf-8')
        print(line.rstrip())
    webpage.close()

main()
```

(All of the roll call votes from 1997 to 2019 are also archived on the book website, if this is more convenient.)

This file is in a format called XML (short for extensible markup language). In XML, data elements are enclosed in matching pairs of *tags*. In this file, all of the data for this particular vote is enclosed in a pair of `<vote-data>` ⋯ `</vote-data>` tags, beginning on line 56. The lines at the beginning of the file are metadata that store information like the name and date of the vote. Between the `<vote-data>` ⋯ `</vote-data>` tags, each vote is enclosed in a pair of `<recorded-vote>` ⋯ `</recorded-vote>` tags, like these (shortened considerably):

```
<recorded-vote><legislator name-id="A000022" ... </recorded-vote>
<recorded-vote><legislator name-id="A000055" ... </recorded-vote>
<recorded-vote><legislator name-id="A000364" ... </recorded-vote>
```

Look carefully at the file to find where the actual votes are cast within these lines. Then write a function

```
partyLine(year, number)
```

that determines whether House roll call vote `number` from the given `year` was a party line vote. Use only the recorded vote elements, and no other parts of the XML files. Do not use any built-in string methods. Be aware that some votes are recorded as Yea/Nay and some are recorded as Aye/No. Do not count votes that are recorded in any other way (e.g., "Present" or "Not Voting"), even toward the total numbers of votes. Test your function thoroughly before moving to the next step.

Question 6.1.1 *What type of data should this function return? (Think about making your life easier for the next function.)*

Question 6.1.2 *Was the vote on the Affordable Care Act a party line vote?*

Question 6.1.3 *Choose another vote that you care about. Was it a party line vote?*

Second, write another function

```
countPartyLine(year, maxNumber)
```

that uses the `partyLine` function to return the fraction of votes that were party line votes during the given `year`. The parameter `maxNumber` is the number of the last roll call vote of the year.

Finally, write a function

```
plotPartyLine()
```

that plots the fractions of party line votes for the last 20 years. To keep things simple, you may assume that there were 450 roll call votes each year. If you would prefer to count all of the roll call votes, here are the numbers for the last several years:

Year	Number	Year	Number
2019	701	2006	543
2018	500	2005	671
2017	710	2004	544
2016	622	2003	677
2015	705	2002	484
2014	564	2001	512
2013	641	2000	603
2012	659	1999	611
2011	949	1998	547
2010	664	1997	640
2009	991	1996	455
2008	690	1995	885
2007	1186	1994	507

Question 6.1.4 *What fraction of votes from each of the last twenty years went along party lines?*

Note that collecting this data may take a long time, so make sure that your functions are correct for a single year first.

Question 6.1.5 *Describe your plot. Has American politics become more polarized over the last 20 years?*

Question 6.1.6 *Many news outlets report on the issue of polarization. Find a news story about this topic online, and compare your results to the story. It might be helpful to think about the motivations of news outlets.*

Part 2: Blue state, red state

We can use the roll call vote files to infer other statistics as well. For this part, write a function

```
stateDivide(state)
```

that plots the number of Democratic and Republican representatives for the given `state` every year for the last 20 years. Your plot should have two different curves (one for Democrat and one for Republican). You may also include curves for other political parties if you would like. Test your function on several different states.

Call your `stateDivide` function for at least five states, including your home state.

Question 6.1.7 *Describe the curves you get for each state. Interpret your results in light of the results from Part 1. Has the House of Representatives become more polarized?*

*Project 6.2 Finding genes

This project assumes that you have read Section 6.8.

In prokaryotic organisms, the coding regions of genes are usually preceded by the start codon ATG, and terminated by one of three stop codons, TAA, TAG, and TGA. A long region of DNA that is framed by start and stop codons, with no stop codons in between, is called an *open reading frame* (or ORF). Searching for sufficiently long ORFs is an effective (but not fool-proof) way to locate putative genes.

For example, the rectangle below marks a particular open reading frame beginning with the start codon ATG and ending with the stop codon TAA. Notice that there are no other stop codons in between.

ggcgg**atg**aaacgcattagcaccaccattaccaccaccatcaccattaccacagg**taa**gttc

Not every ORF corresponds to a gene. Since there are $4^3 = 64$ possible codons and 3 different stop codons, one is likely to find a stop codon approximately every $64/3 \approx 21$ codons in a random stretch of DNA. Therefore, we are really only interested in ORFs that have lengths substantially longer than this. Since such long ORFs are unlikely to occur by chance, there is good reason to believe that they may represent a coding region of a gene.

Not all open reading frames on a strand of DNA will be aligned with the left (5') end of the sequence. For instance, in the example above, if we only started searching for open reading frames in the codons aligned with the left end — `ggc`, `gga`, `tga`, etc. — we would have missed the boxed open reading frame. To find all open reading frames in a strand of DNA, we must look at the codon sequences with offsets 0, 1, and 2 from the left end. (See Exercise 6.8.6 for another example.) The codon sequences with these offsets are known as the three *forward reading frames*. In the example above, the three forward reading frames begin as follows:

Reading frame 0: `ggc gga tga aac` ...

Reading frame 1: `gcg gat gaa acg` ...

Reading frame 2: `cgg atg aaa cgc` ...

Because DNA is double stranded, with a reverse complement sequence on the other strand, there are also three *reverse reading frames*. In this strand of DNA, the reverse reading frames would be:

Reverse reading frame 0: `gaa ctt acc tgt` ...

Reverse reading frame 1: `aac tta cct gtg` ...

Reverse reading frame 2: `act tac ctg tgg` ...

Because genomic sequences are so long, finding ORFs must be performed computationally. For example, consider the following sequence of 1,260 nucleotides from an *Escherichia coli* (or *E. coli*) genome, representing only about 0.03% of the complete

genome. (This number represents about 0.00005% of the human genome.) Can you find an open reading frame in this sequence?

```
agcttttcattctgactgcaacgggcaatatgtctctgtgtggattaaaaaaagagtgtctgatagcagc
ttctgaactggttacctgccgtgagtaaattaaaatttttattgacttaggtcactaaatactttaaccaa
tataggcatagcgcacagacagataaaaattacagagtacacaacatccatgaaacgcattagcaccacc
attaccaccaccatcaccattaccacaggtaacggtgcgggctgacgcgtacaggaaacacagaaaaaag
cccgcacctgacagtgcgggctttttttttcgaccaaaggtaacgaggtaacaaccatgcgagtgttgaag
ttcggcggtacatcagtggcaaatgcagaacgttttctgcgggttgccgatattctggaaagcaatgcca
ggcaggggcaggtggccaccgtcctctctgcccccgccaaaatcaccaaccatctggtagcgatgattga
aaaaaccattagcggtcaggatgctttacccaatatcagcgatgccgaacgtatttttgccgaacttctg
acgggactcgccgccgcccagccgggatttccgctggcacaattgaaaactttcgtcgaccaggaatttg
cccaaataaaacatgtcctgcatggcatcagtttgttggggcagtgcccggatagcatcaacgctgcgct
gatttgccgtggcgagaaaatgtcgatcgccattatggccggcgtgttagaagcgcgtggtcacaacgtt
accgttatcgatccggtcgaaaaactgctggcagtgggtcattacctcgaatctaccgttgatattgctg
aatccacccgccgtattgcggcaagccgcattccggctgaccacatggtgctgatggctggtttcactgc
cggtaatgaaaaaggcgagctggtggttctgggacgcaacggttccgactactccgctgcggtgctggcg
gcctgtttacgcgccgattgttgcgagatctggacggatgttgacggtgtttatacctgcgatccgcgtc
aggtgcccgatgcgaggttgttgaagtcgatgtcctatcaggaagcgatggagctttcttacttcggcgc
taaagttcttcacccccgcaccattacccccatcgcccagttccagatcccttgcctgattaaaaatacc
ggaaatccccaagcaccaggtacgctcattggtgccagccgtgatgaagacgaattaccggtcaagggca
```

This is clearly not a job for human beings. But, if you spent enough time, you might spot an ORF in reading frame 2 between positions 29 and 97, an ORF in reading frame 0 between positions 189 and 254, and another ORF in reading frame 0 between positions 915 and 1073 (to name just a few). These are highlighted in red below.

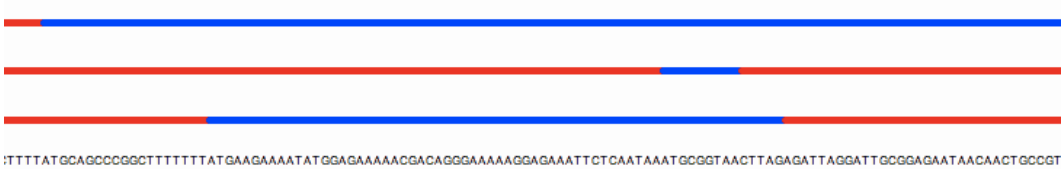
```
agcttttcattctgactgcaacgggcaatATGTCTCTGTGTGGATTAAAAAAAGAGTGTCTGATAGCAGC
TTCTGAACTGGTTACCTGCCGTGAGTAAattaaaatttttattgacttaggtcactaaatactttaaccaa
tataggcatagcgcacagacagataaaaattacagagtacacaacatccATGAAACGCATTAGCACCACC
ATTACCACCACCATCACCATTACCACAGGTAACGGTGCGGGCTGAcgcgtacaggaaacacagaaaaaag
cccgcacctgacagtgcgggctttttttttcgaccaaaggtaacgaggtaacaaccatgcgagtgttgaag
ttcggcggtacatcagtggcaaatgcagaacgttttctgcgggttgccgatattctggaaagcaatgcca
ggcaggggcaggtggccaccgtcctctctgcccccgccaaaatcaccaaccatctggtagcgatgattga
aaaaaccattagcggtcaggatgctttacccaatatcagcgatgccgaacgtatttttgccgaacttctg
acgggactcgccgccgcccagccgggatttccgctggcacaattgaaaactttcgtcgaccaggaatttg
cccaaataaaacatgtcctgcatggcatcagtttgttggggcagtgcccggatagcatcaacgctgcgct
gatttgccgtggcgagaaaatgtcgatcgccattatggccggcgtgttagaagcgcgtggtcacaacgtt
accgttatcgatccggtcgaaaaactgctggcagtgggtcattacctcgaatctaccgttgatattgctg
aatccacccgccgtattgcggcaagccgcattccggctgaccacatggtgctgatggctggtttcactgc
cggtaATGAAAAAGGCGAGCTGGTGGTTCTGGGACGCAACGGTTCCGACTACTCCGCTGCGGTGCTGGCG
GCCTGTTTACGCGCCGATTGTTGCGAGATCTGGACGGATGTTGACGGTGTTTATACCTGCGATCCGCGTC
AGGTGCCCGATGCGAGGTTGTTGAagtcgatgtcctatcaggaagcgatggagctttcttacttcggcgc
taaagttcttcacccccgcaccattacccccatcgcccagttccagatcccttgcctgattaaaaatacc
ggaaatccccaagcaccaggtacgctcattggtgccagccgtgatgaagacgaattaccggtcaagggca
```

Of these three, only the second is actually a known gene.

Part 1: Find ORFs

For the first part of this project, we will read a text file containing a long DNA sequence, write this sequence to a turtle graphics window, and then draw bars with

intervals of alternating colors over the sequence to indicate the locations of open reading frames. In the image below, which represents just a portion of a window, the three bars represent the three forward reading frames with reading frame 0 at the bottom. Look closely to see how the blue bars begin and end on start and stop codons (except for the top bar which ends further to the right). The two shorter blue bars are actually unlikely to be real genes due to their short length.



To help you get started and organize your project, you can download a "skeleton" of the program from the book website. In the program, the `viewer` function sets up the turtle graphics window, writes the DNA sequence at the bottom (one character per x coordinate), and then calls the two functions that you will write. The `main` function reads a long DNA sequence from a file and into a string variable, and then calls `viewer`.

To display the open reading frames, you will write the function

```
orf1(dna, rf, tortoise)
```

to draw colored bars representing open reading frames in one forward reading frame with offset `rf` (0, 1, or 2) of string `dna` using the turtle named `tortoise`. This function will be called three times with different values of `rf` to draw the three reading frames. As explained in the skeleton program, the drawing function is already written; you just have to change colors at the appropriate times before calling it. Hint: Use a Boolean variable `inORF` in your `for` loop to keep track of whether you are currently in an ORF.

Also on the book site are files containing various size prefixes of the genome of a particular strain of *E. coli*. Start with the smaller files.

Question 6.2.1 *How long do you think an open reading frame should be for us to consider it a likely gene?*

Question 6.2.2 *Where are the likely genes in the first 5,000 bases of the E. coli genome?*

Part 2: GC content

The GC content of a particular stretch of DNA is the ratio of the number of C and G bases to the total number of bases. For example, in the following sequence

```
TCTACGACGT
```

the GC content is 0.5 because 5/10 of the bases are either C or G. Because the GC content is usually higher around coding sequences, this can also give clues about gene location. (This is actually more true in vertebrates than in bacteria like *E. coli*.) In long sequences, GC content can be measured over "windows" of a fixed size.

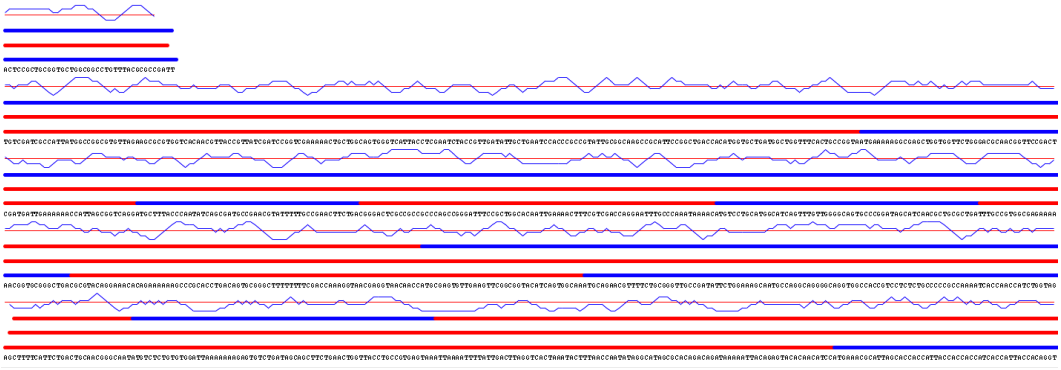


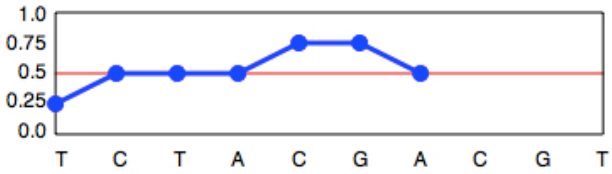
Figure 1 The finished product.

For example, in the tiny example above, if we measure GC content over windows of size 4, the resulting ratios are

```
TCTACGACGT

TCTA          0.25
 CTAC         0.50
  TACG        0.50
   ACGA       0.50
    CGAC      0.75
     GACG     0.75
      ACGT    0.50
```

We can then plot this like so:



Finish writing the function

```
gcFreq(dna, window, tortoise)
```

that plots the GC frequency of the string `dna` over windows of size `window` using the turtle named `tortoise`. Plot this in the same window as the ORF bars. As explained in the skeleton code, the plotting function is already written; you just need to compute the correct GC fractions. You should not need to count the GC content anew for each window. Once you have counted the G and C bases for the first window, you can incrementally modify this count for the subsequent windows. The final display should look something like Figure 1.