Find Unclustered Proteins

Run `make data` to get the data you need for this exercise or manually download the data:

```
wget ftp://ftp.imicrobe.us/biosys-analytics/exercises/unclustered-proteins.tgz
```

Unpack the tarball with `tar xvf unclustered-proteins.tgz`.

Write a Python program called `find_unclustered.py` that will create a FASTA file of the unclustered proteins. The parameters to the program should be:

- `-p|--proteins`: name of the FASTA file containing the original protein sequences given to CD-HIT to cluster
- `-c|--cdhit`: name of the CD-HIT cluster file
- `-o|--outfile` argument (default `unclustered.fa`) where to write the sequences.

```
$ ./find_unclustered.py -h
usage: find_unclustered.py [-h] -c cdhit -p proteins [-o outfile]

Find proteins not clustered by CD-HIT

optional arguments:
  -h, --help            show this help message and exit
  -c cdhit, --cdhit cdhit
                        Output file from CD-HIT (clustered proteins) (default:
                        None)
  -p proteins, --proteins proteins
                        Proteins FASTA (default: None)
  -o outfile, --outfile outfile
                        Output file (default: unclustered.fa)
```

If successful, report the number of unclustered proteins written to the indicated output file:

```
$ ./find_unclustered.py -p unclustered-proteins/proteins.fa \
  -c unclustered-proteins/proteins.fa.cdhit.clstr
Wrote 309 of 220,520 unclustered proteins to "unclustered.fa"
```

Parsing the cluster file

The file `unclustered-proteins/proteins.fa.cdhit.clstr` contains the protein IDs that were clustered and put into HMM profiles. It is almost in FASTA format, but not quite:

```
$ head unclustered-proteins/proteins.fa.cdhit.clstr
>Cluster 0
0   7391aa, >444908222|dbj|BAM78286.1|... *
1   7391aa, >444908224|dbj|BAM78287.1|... at 94.13%
2   7391aa, >449147289|ref|YP_007438864.1|... at 100.00%
3   555aa, >461495565|dbj|BAM99887.1|... at 99.10%
4   555aa, >461495567|dbj|BAM99888.1|... at 99.10%
5   555aa, >461495569|dbj|BAM99889.1|... at 99.28%
6   555aa, >461495571|dbj|BAM99890.1|... at 99.28%
7   555aa, >461495573|dbj|BAM99891.1|... at 99.10%
8   555aa, >461495575|dbj|BAM99892.1|... at 99.28%
```

We cannot use `SeqIO.parse()` to read this file. We'll have to parse it manually:

```
for line in args.cdhit:
    ....
```

The names of the clusters, e.g., "Cluster 0," start with `>`, so we want to skip those lines. How can you identify a `line` that *starts with* `>`? When you find one, how can you skip to the next iteration of the loop?

The lines you want are the *other* lines that list the proteins that were included in the cluster. The protein IDs are in the bit that looks like ">444908222|dbj|BAM78286.1|..." where the ID is "444908222". A regex would be the perfect thing to extract this:

```
match = re.search(r'>(\d+)', line)
```

- The `r''` makes a "raw" string so that Python doesn't try to interpret the `\d` like it would `\n` for a newline. Here the `\d` needs to be passed *literally* to the regex engine.
- The `>` is a literal character.
- The `\d` is a shortcut pattern that matches any **d**igit and is the same as the character class `[0-9]` which is the same as `[01234565789]`.
- The `+` makes it match *one or more*.
- The parentheses around the regex will cause the regex engine to *capture* the text that matches so we can retrieve it later.

If `re.search()` fails to match, it will return `None`; otherwise it matched:

```
>>> import re
>>> line = '0   7391aa, >444908222|dbj|BAM78286.1|... *'
>>> match = re.search(r'>(\d+)', line)
>>> match
<re.Match object; span=(12, 22), match='>444908222'>
```

The `match.group()` function will return the value of the text that was capturedf by the engine. The groups are numbered by the position of their opening paren:

```
>>> match.group(1)
'444908222'
```

Noting the clustered IDs

We will need to remember all the protein IDs in some sort of data structure thatwe can use to look up. Given that a protein might be clustered more than once, a `list` is not optimal as it will allow for duplications. Also the time to search a `list` grows linearly.

We've used a `dict` as a lookup, and that could work here. We don't actually care about the value, we just want to store a unique key which is the protein ID:

```
>>> id = match.group(1)
>>> id
'444908222'
>>> protein_ids = {}
>>> protein_ids[id] = 1
```

A better option is a `set` which is like a unique list or like just the keys of a dictionary:

```
>>> protein_ids = set()
>>> protein_ids.add(id)
>>> protein_ids
{'444908222'}
```

Just as with a `list` or a `dict`, you can use `x in y` to see if a given protein ID is in the set of protein IDs:

```
>>> '444908222' in protein_ids
True
```

Parsing the proteins FASTA

The file `proteins.fa` is a normal FASTA file, and you can use `SeqIO.parse()` to handle it. Each record returned by this function is a *object* representing all the details of a FASTA record:

```
>>> for rec in SeqIO.parse('proteins.fa', 'fasta'):
...     print(rec)
...     break
...
ID: 388548806
Name: 388548806
Description: 388548806
Number of features: 0
Seq('MDLQSLVAEVASLRAQLQQLQSQPGKKSPCQGVTGKGTPCRNGALVGAPYCRMH...ESV',
SingleLetterAlphabet())
```

You want to look at the `rec.id` for the ID. It's usually the only thing there:

```
$ grep '>' proteins.fa | head -5
>388548806
>388548807
>388548808
>388548809
>388548810
```

But not always:

```
$ grep -e '^>' proteins.fa | sed "s/^>//" | grep -v -P '^\d+$' | head -5
26788002|emb|CAD19173.1| putative RNA helicase, partial [Agaricus bisporus virus X]
26788000|emb|CAD19172.1| putative RNA helicase, partial [Agaricus bisporus virus X]
985757046|ref|YP_009222010.1| hypothetical protein [Alternaria brassicicola
fusarivirus 1]
985757045|ref|YP_009222011.1| hypothetical protein [Alternaria brassicicola
fusarivirus 1]
985757044|ref|YP_009222009.1| polyprotein [Alternaria brassicicola fusarivirus 1]
```

So you will need to remove any characters starting with `|`. How will you do that? You might use `str.split()`, or you could consider using `re.sub()` which *substitutes* any matching regex pattern for another value in a string. This *returns a new string*:

```
>>> import re
>>> id = '26788002|emb|CAD19173.1| putative RNA helicase, partial'
>>> re.sub(r'\|.*', '', id)
'26788002'
```

Finding the unclustered IDs

Once you have parsed the unclustered IDs into a `set`, you will parse the `proteins.fa` and find the protein IDs. If a given protein ID was not in the set of clustered IDs, you will write it to the output file:

```
for rec in SeqIO.parse(args.proteins, 'fasta'):
    prot_id = ... ①
    if prot_id not in ...: ②
        SeqIO.write(...) ③
```

① How will you get the clean protein ID?

② This is how you will determine if a given ID is `in` the clustered IDs.

③ If the ID was not clustered, you will write it to the output file.

Test

A passing test suite looks like this:

```
$ make test
pytest --disable-pytest-warnings -xv test.py
============================= test session starts ==============================
...
collected 8 items

test.py::test_exists PASSED                                              [ 12%]
test.py::test_usage PASSED                                               [ 25%]
test.py::test_missing_cdhit PASSED                                       [ 37%]
test.py::test_missing_proteins PASSED                                    [ 50%]
test.py::test_bad_protein_file PASSED                                    [ 62%]
test.py::test_bad_cdhit_file PASSED                                      [ 75%]
test.py::test_good_input PASSED                                          [ 87%]
test.py::test_outfile PASSED                                             [100%]

========================= 8 passed, 1 warning in 5.79s =========================
```