# Programming Homework 1

Help Center

The **due date** for this homework is **Sun 11 Oct 2015 6:30 PM CDT**.

In lecture and in a practical, we saw an implementation of the naive exact matching algorithm:

```
def naive(p, t):
    occurrences = []
    for i in range(len(t) - len(p) + 1):  # loop over alignments
        match = True
        for j in range(len(p)):  # loop over characters
            if t[i+j] != p[j]:  # compare characters
                match = False
                break
        if match:
            occurrences.append(i)  # all chars matched; record
    return occurrences
```

...and we saw a function that takes a DNA string and returns its reverse complement:

```
def reverseComplement(s):
    complement = {'A': 'T', 'C': 'G', 'G': 'C', 'T': 'A', 'N': 'N'}
    t = ''
    for base in s:
        t = complement[base] + t
    return t
```

...and we saw a function that parses a DNA reference genome from a file in the FASTA format.

```
def readGenome(filename):
    genome = ''
    with open(filename, 'r') as f:
        for line in f:
            # ignore header line with genome information
            if not line[0] == '>':
                genome += line.rstrip()
```

```
    return genome
```

...and we saw a function that parses the read and quality strings from a FASTQ file containing sequencing reads.

```
def readFastq(filename):
    sequences = []
    qualities = []
    with open(filename) as fh:
        while True:
            fh.readline()  # skip name line
            seq = fh.readline().rstrip()  # read base sequence
            fh.readline()  # skip placeholder line
            qual = fh.readline().rstrip() # base quality line
            if len(seq) == 0:
                break
            sequences.append(seq)
            qualities.append(qual)
    return sequences, qualities
```

First, implement a version of the naive exact matching algorithm that is *strand-aware*. That is, instead of looking only for occurrences of P in T, additionally look for occurrences of the *reverse complement* of P in T. If P is `ACT`, your function should find occurrences of both `ACT` and its reverse complement `AGT` in T.

If P and its reverse complement are identical (e.g. `AACGTT`), then a given match offset should be reported only once. So if your new function is called `naive_with_rc`, then the old `naive` function and your new `naive_with_rc` function should return the same results when P equals its reverse complement.

Hint: See this notebook for a few examples you can use to test your `naive_with_rc` function.

Next, download and parse the lambda virus genome, at:

https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/lambda_virus.fa

☑ **In accordance with the Coursera Honor Code, I (Oanh) certify that the answers here are my own work.** **Thank you!**

# Question 1

How many times does `AGGT` or its reverse complement (`ACCT`) occur in the lambda virus genome? E.g. if `AGGT` occurs 10 times and `ACCT` occurs 12 times, you should report 22.

2888

# Question 2

How many times does `TTAA` or its reverse complement occur in the lambda virus genome? Hint: `TTAA` and its reverse complement are equal, so remember not to double count.

195

# Question 3

What is the offset of the leftmost occurrence of `ACTAAGT` or its reverse complement in the Lambda virus genome? E.g. if the leftmost occurrence of `ACTAAGT` is at offset 40 (0-based) and the leftmost occurrence of its reverse complement `ACTTAGT` is at offset 29, then report 29.

430

# Question 4

What is the offset of the leftmost occurrence of `AGTCGA` or its reverse complement in the Lambda virus genome?

8

8

## Question 5

As we will discuss, sometimes we would like to find *approximate* matches for *P* in *T*. That is, we want to find occurrences with one or more differences.

For Questions 5 and 6, make a new version of the `naive` function called `naive_2mm` that allows up to 2 mismatches per occurrence. Unlike for the previous questions, **do not consider the reverse complement here**. We're looking for approximate matches for *P* itself, not its reverse complement.

For example, `ACTTTA` occurs twice in `ACTTACTTGATAAAGT`, once at offset 0 with 2 mismatches, and once at offset 4 with 1 mismatch. So `naive_2mm('ACTTTA', 'ACTTACTTGATAAAGT')` should return the list `[0, 4]`.

Hint: See this notebook for a few examples you can use to test your `naive_2mm` function.

How many times does `TTCAAGCC` occur in the Lambda virus genome when allowing up to 2 mismatches?

191

## Question 6

What is the offset of the leftmost occurrence of `AGGAGGTT` in the Lambda virus genome when allowing up to 2 mismatches?

49

# Question 7

Finally, download and parse the provided FASTQ file containing real DNA sequencing reads derived from a human:

https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/ERR037900_1.first1000.fastq

Note that the file has *many* reads in it and you should examine all of them together when answering this question. The reads are taken from this study:

Ajay, S. S., Parker, S. C., Abaan, H. O., Fajardo, K. V. F., & Margulies, E. H. (2011). Accurate and comprehensive sequencing of personal genomes. *Genome research*, 21(9), 1498-1505.

This dataset has something wrong with it; one of the sequencing cycles is poor quality.

Report which sequencing cycle has the problem. Remember that a sequencing cycle corresponds to a particular offset in *all* the reads. For example, if the leftmost read position seems to have a problem consistently across reads, report **0**. If the fourth position from the left has the problem, report **3**. Do whatever analysis you think is needed to identify the bad cycle. It might help to review the "Analyzing reads by position" video.

65

**In accordance with the Coursera Honor Code, I (Oanh) certify that the answers here are my own work.** **Thank you!**

Submit Answers Save Answers