**Programming Homework 3**

The due date for this homework is Sun 27 Sep 2015 6:30 PM CDT.

We saw how to adapt dynamic programming to find approximate occurrences of a pattern in a text. Recall that:

Rows of the dynamic programming matrix are labeled with bases from P and columns with bases from T Elements in the first row are set to 0 Elements in the first column are set to 0, 1, 2, ..., as for edit distance Other elements are set in the same way as elements of a standard edit distance matrix The minimal value in the bottom row is the edit distance of the closest match between P and T First, download the provided excerpt of human chromosome 1

<https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/chr1.GRCh38.excerpt.fasta>

Second, parse it using the readGenome function we wrote before.

Third, adapt the editDistance function we saw in practical (copied below) to answer questions 1 and 2 below. Your function should take arguments p (pattern), t (text) and should return the edit distance of the match between P and T with the fewest edits.

In [1]:

**def** loadFasta(filename):

f = open(filename)

name = ''

dna = ''

**for** line **in** f:

**if** line[0] == '>':

name = line[1:].rstrip()

**else**:

dna += line.rstrip()

**return** name, dna

In [2]:

chromosome\_name, chromosome\_dna = loadFasta('chr1.GRCh38.excerpt.fasta')

In [3]:

**def** editDistance(x, y):

*# Create distance matrix*

D = []

**for** i **in** range(len(x)+1):

D.append([0]\*(len(y)+1))

*# Initialize first row and column of matrix*

**for** i **in** range(len(x)+1):

D[i][0] = i

**for** i **in** range(len(y)+1):

D[0][i] = 0

*# Fill in the rest of the matrix*

**for** i **in** range(1, len(x)+1):

**for** j **in** range(1, len(y)+1):

distHor = D[i][j-1] + 1

distVer = D[i-1][j] + 1

**if** x[i-1] == y[j-1]:

distDiag = D[i-1][j-1]

**else**:

distDiag = D[i-1][j-1] + 1

D[i][j] = min(distHor, distVer, distDiag)

*# Edit distance is the value in the bottom right corner of the matrix*

distance = D[-1]

distance.sort()

**return** distance[0]

**Question 1**

What is the edit distance of the best match between pattern GCTGATCGATCGTACG and the excerpt of human chromosome 1? (Don't consider reverse complements.)

In [4]:

p = 'GCTGATCGATCGTACG'

In [5]:

%%**time**

editDistance(p, chromosome\_dna)

Wall time: 13.4 s

Out[5]:

3

**Answer:**

3

**Question 2**

What is the edit distance of the best match between pattern GATTTACCAGATTGAG and the excerpt of human chromosome 1? (Don't consider reverse complements.)

In [6]:

p = 'GATTTACCAGATTGAG'

In [7]:

%%**time**

editDistance(p, chromosome\_dna)

Wall time: 12.9 s

Out[7]:

2

**Answer:**

2

**Question 3**

In a practical, we saw a function for finding the longest exact overlap (suffix/prefix match) between two strings. The function is copied below.

In [8]:

**def** overlap(a, b, min\_length=3):

*""" Return length of longest suffix of 'a' matching*

*a prefix of 'b' that is at least 'min\_length'*

*characters long. If no such overlap exists,*

*return 0. """*

start = 0 *# start all the way at the left*

**while** **True**:

start = a.find(b[:min\_length], start) *# look for b's prefix in a*

**if** start == -1: *# no more occurrences to right*

**return** 0

*# found occurrence; check for full suffix/prefix match*

**if** b.startswith(a[start:]):

**return** len(a)-start

start += 1 *# move just past previous match*

In [9]:

**from** **itertools** **import** permutations

**def** overlap(a, b, min\_length=3):

*""" Return length of longest suffix of 'a' matching*

*a prefix of 'b' that is at least 'min\_length'*

*characters long. If no such overlap exists,*

*return 0. """*

start = 0 *# start all the way at the left*

**while** **True**:

start = a.find(b[:min\_length], start) *# look for b's prefix in a*

**if** start == -1: *# no more occurrences to right*

**return** 0

*# found occurrence; check for full suffix/prefix match*

**if** b.startswith(a[start:]):

**return** len(a)-start

start += 1 *# move just past previous match*

Say we are concerned only with overlaps that (a) are exact matches (no differences allowed), and (b) are at least k bases long. To make an overlap graph, we could call overlap(a, b, min\_length=k) on every possible pair of reads from the dataset. Unfortunately, that will be very slow!

Consider this: Say we are using k=6, and we have a read a whose length-6 suffix is GTCCTA. Say GTCCTA does not occur in any other read in the dataset. In other words, the 6-mer GTCCTA occurs at the end of read a and nowhere else. It follows that a's suffix cannot possibly overlap the prefix of any other read by 6 or more characters.

Put another way, if we want to find the overlaps involving a suffix of read a and a prefix of some other read, we can ignore any reads that don't contain the length-k suffix of a. This is good news because it can save us a lot of work!

Here is a suggestion for how to implement this idea. You don't have to do it this way, but this might help you. Let every k-mer in the dataset have an associated Python set object, which starts out empty. We use a Python dictionary to associate each k-mer with its corresponding set. (1) For every k-mer in a read, we add the read to the set object corresponding to that k-mer. If our read is GATTA and k=3, we would add GATTA to the set objects for GAT, ATT and TTA. We do this for every read so that, at the end, each set contains all reads containing the corresponding k-mer. (2) Now, for each read a, we find all overlaps involving a suffix of a. To do this, we take a's length-k suffix, find all reads containing that k-mer (obtained from the corresponding set) and call overlap(a, b, min\_length=k) for each.

The most important point is that we do not call overlap(a, b, min\_length=k) if b does not contain the length-k suffix of a.

Download and parse the read sequences from the provided Phi-X FASTQ file. We'll just use their base sequences, so you can ignore read names and base qualities. Also, no two reads in the FASTQ have the same sequence of bases. This makes things simpler.

<https://d28rh4a8wq0iu5.cloudfront.net/ads1/data/ERR266411_1.for_asm.fastq>

Next, find all pairs of reads with an exact suffix/prefix match of length at least 30. Don't overlap a read with itself; if a read has a suffix/prefix match to itself, ignore that match. Ignore reverse complements.

Hint 1: Your function should not take much more than 15 seconds to run on this 10,000-read dataset, and maybe much less than that. (Our solution takes about 3 seconds.) If your function is much slower, there is a problem somewhere. Hint 2: Remember not to overlap a read with itself. If you do, your answers will be too high. Hint 3: You can test your implementation by making up small examples, then checking that (a) your implementation runs quickly, and (b) you get the same answer as if you had simply called overlap(a, b, min\_length=k) on every pair of reads. We also have provided a couple examples you can check against. Picture the overlap graph corresponding to the overlaps just calculated. How many edges are in the graph? In other words, how many distinct pairs of reads overlap?

In [10]:

**def** naive\_overlap\_map(reads, k):

olaps = {}

**for** a, b **in** permutations(reads, 2):

olen = overlap(a, b, min\_length=k)

**if** olen > 0:

olaps[(a, b)] = olen

**return** olaps

In [11]:

**def** readFASTQ(filename):

f = open(filename)

reads = []

**while** **True**:

h = f.readline()

**if** len(h) == 0:

**break**

sequence = f.readline().rstrip()

\_ = f.readline()

quality = f.readline().rstrip()

reads.append(sequence)

**return** reads

In [12]:

file = 'ERR266411\_1.for\_asm.fastq'

reads = readFASTQ(file)

In [14]:

%%**time**

overlaps = naive\_overlap\_map(reads, 30)

Wall time: 1min 33s

In [15]:

len(overlaps)

Out[15]:

904746

**Answer:**

904746

In [16]:

s = set()

**for** a, b **in** overlaps.keys():

s.add(a)

In [17]:

len(s)

Out[17]:

7161

**Answer:**

7161

In [3]:

**from** **itertools** **import** permutations

**def** overlap(a, b, min\_length=3):

*""" Return length of longest suffix of 'a' matching*

*a prefix of 'b' that is at least 'min\_length'*

*characters long. If no such overlap exists,*

*return 0. """*

start = 0 *# start all the way at the left*

**while** **True**:

start = a.find(b[:min\_length], start) *# look for b's prefix in a*

**if** start == -1: *# no more occurrences to right*

**return** 0

*# found occurrence; check for full suffix/prefix match*

**if** b.startswith(a[start:]):

**return** len(a)-start

start += 1 *# move just past previous match*

In [4]:

**def** readFASTQ(filename):

f = open(filename)

reads = []

**while** **True**:

h = f.readline()

**if** len(h) == 0:

**break**

sequence = f.readline().rstrip()

\_ = f.readline()

quality = f.readline().rstrip()

reads.append(sequence)

**return** reads

In [41]:

**def** naive\_overlap\_faster(reads, k):

olaps = {}

kmer\_dict = {}

**for** read **in** reads:

**for** i **in** range(len(read)-k+1):

kmer\_dict[read[i:i+k]] = set()

**for** read **in** reads:

**for** i **in** range(len(read)-k+1):

kmer\_dict[read[i:i+k]].add(read)

effective\_read = []

**for** read **in** reads:

current\_kmer\_set = kmer\_dict[read[-1\*k:]]

**for** kmer\_read **in** current\_kmer\_set:

**if** read != kmer\_read:

olen = overlap(read, kmer\_read, min\_length=k)

**if** olen > 0:

olaps[(read, kmer\_read)] = olen

olen = 0

**return** olaps

In [42]:

file = 'ERR266411\_1.for\_asm.fastq'

reads = readFASTQ(file)

In [44]:

%%**time**

len(naive\_overlap\_faster(reads, 30))

Wall time: 4.35 s

Out[44]:

904746

In [19]:

len(reads)

Out[19]:

10000