Genomics
Exercise 2

Read Mapping Considerations
Hamming and Edit Distance

EXERCISE 2.1
Exercise 2.1 a)

a) Write up the definition of the hamming distance and edit distance. For the edit distance, consider only *insert*, *delete*, and *replace* operations. Each operation has a cost of one.

**Hamming Distance**

**Definition:** An alphabet $\Sigma$ is a finite set of characters, e.g. $\Sigma = \{C, G, A, T\}$ for DNA.

**Definition:** A sequence $S$ of length $n$ ($0 \leq n$) over an alphabet $\Sigma$ is an ordered list of characters from $\Sigma$. The i-th character from $S$ is denoted as $S[i]$ ($i = 0 \ldots n-1$).

**Definition:** Given two sequences $A$ and $B$ of the same length $n$, the Hamming distance $H(A, B)$ of $A$ and $B$ is defined as $\sum_{i=0}^{n-1} I(A[i], B[i])$ where $I$ is the indicator function (i.e. $I(x, y) = 1$ if $x = y$ and $I(x, y) = 0$ otherwise).

**Observation:** $H(A, B) \leq n$
Exercise 2.1 a)

a) Write up the definition of the hamming distance and edit distance. For the edit distance, consider only insert, delete, and replace operations. Each operation has a cost of one.

**Edit Distance**

**Definition:** Given a sequence $S$ of length $n$, an integer $i$ ($0 \leq i \leq n$), and a character $x \in \sum$, we define the insert operation $\text{ins}(S, i, x)$ as a function $\text{ins} : \sum^n \rightarrow \sum^{n+1}$ that inserts $x$ into $S$ after $S[i]$.

**Definition:** Given a sequence $S$ of length $n$ ($1 \leq n$) and an integer $i$ ($0 \leq i \leq n - 1$), we define the delete operation $\text{del}(S, i, x)$ as a function $\text{del} : \sum^n \rightarrow \sum^{n-1}$ that removes $S[i]$ from $S$.

**Definition:** Given a sequence $S$ of length $n$, an integer $i$ ($0 \leq i \leq n - 1$), and a character $x \in \sum$, we define the replace operation $\text{rep}(S, i, x)$ as a function $\text{rep} : \sum^n \rightarrow \sum^n$ that replaces $S[i]$ by $x$.

**Definition:** Given two sequences $A$ and $B$ of lengths $n$ and $m$, we define the edit distance $E(A, B)$ as the smallest number of insertion, deletion, and replacement operations such that after applying the operations, $A$ is transformed into $B$.

**Observation:** $E(A, B) \leq \max(n, m)$
b) Determine the Hamming distances and the edit distance between the following pairs of strings.

**Hamming Distance (counting mismatches)**

<table>
<thead>
<tr>
<th></th>
<th>TGGTACTTCTC</th>
<th>TGGTGGTGG</th>
<th>TAGGTGGTGG</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

X = 3                                                = 0                                                = 5

|       | TAGTTCTTCTT | TGGTGGTGG | TGGTGGTGG |

**Edit Distance (alignments)**

<table>
<thead>
<tr>
<th></th>
<th>TGGTACTTCTC</th>
<th>TGGTGGTGG</th>
<th>TAGGTGGTGG-</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TGGTACTTCTC</td>
<td>TGGTGGTGG</td>
<td>TGGTGGTGG-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TGGTACTTCTC</th>
<th>TGGTGGTGG</th>
<th>TAGGTGGTGG-</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TGGTACTTCTC</td>
<td>TGGTGGTGG</td>
<td>TGGTGGTGG-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>TGGTACTTCTC</th>
<th>TGGTGGTGG</th>
<th>TAGGTGGTGG-</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TGGTACTTCTC</td>
<td>TGGTGGTGG</td>
<td>TGGTGGTGG-</td>
</tr>
</tbody>
</table>
Exercise 2.1 c)

c) Write a function, method, or program in a programming language of your choice to determine the edit distance between two given text strings. Test it with the examples of b)

```python
13:27:48 ~ $ python
Python 2.7.5+ (default, Sep 19 2013, 13:48:49)
[GCC 4.8.1] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> # Adapted from http://en.wikibooks.org/wiki/Algorithm_Implementation
... def edit_distance(s1, s2):
...    # Handle corner cases, below we can assume len(s1) >= len(s2) > 0.
...    if len(s1) < len(s2): return edit_distance(s2, s1)
...    if len(s2) == 0: return len(s1)
...    # We fill the matrix column-wise.
...    previous_col = xrange(len(s2) + 1)  # == [0, 1, ..., len(s2)]
...    for i, c1 in enumerate(s1):
...        current_col = [i + 1]
...        for j, c2 in enumerate(s2):
...            hor = previous_col[j + 1] + 1        # horizontal in matrix
...            vert = current_col[j] + 1            # vertical in matrix
...            diag = previous_col[j] + (c1 != c2)  # diagonal in matrix
...            current_col.append(min(vert, hor, diag))
...        previous_col = current_col
...    return previous_col[-1]
```
Exercise 2.1 c)

c) Write a function, method, or program in a programming language of your choice to
determine the edit distance between two given text strings. Test it with the examples
of b)

```python
...     return previous_row[-1]
...
>>> edit_distance('TGGTACTTCTC', 'TAGTTCTTCTT')
3
>>> edit_distance('TGGTGGTGG', 'TGGTGGTGG')
0
>>> edit_distance('TAGGTGGTG', 'TGGTGGTGG')
2
...
Read Mapping

EXERCISE 2.2
Exercise 2.2 a)

a) What is the purpose of read mapping in a next generation sequencing workflow? Which constraints make it special from more general approximate string matching problems?

**Purpose (for whole genome/exome sequencing)**

- Given a reference sequence $S$ and a large set $R$ of short reads $r$ from a donor that has a genome $G$ that is similar to $S$.
- The overall aim of WGS/WES is to measure features of the donor’s genome (e.g. SNPs, small indels, structural variants, copy number variations, …).
- Ideally, we want to find the positions in $S$ for each $r$ that correspond to the sample positions in $G$.
- Many practitioners want to find a (the?) position in $S$ that corresponds to the sample position in $G$ in the “best” fashion.
- Another option would be to enumerate a set of positions in $S$ that are likely to correspond to the sample position of $r$ in $G$. After mapping all reads, a post-processing step could be used to select a “best” location for each read using the “global” view. However, this is rarely (if ever) done given the huge data sets generated by NGS.
Exercise 2.2 b)

b) Give a formal definition of the read mapping problem.

For Hamming distance

Given a reference sequence S over an alphabet $\Sigma$, a set R of reads r, the Hamming distance function H, and a maximal distance k.

For each read r, we now want to find a set of matches (locations) in S.

A feasible match is a match with distance $\leq k$. A best match for r is a feasible match that has the smallest distance of all feasible matches. There can be more than one best match. Matches can be ranked ascendingly by their distance.

We now can define multiple problems. For each read (1) find a best match, (2) find all best matches, (3) find up to c best matches (for a constant c), (4) find up to c best-ranking feasible matches, (5) find all matches, ...

The extension to forward and reverse strand of the reference is trivial.

For Edit distance

The definition of a match becomes more complicated (see lecture script and Holtgrewe et al., 2011) but the remaining definition remains the same.
Exercise 2.2 c)

c) Solve the following read mapping problem instances. The distance function is the edit distance. All reads are of good quality. Write down all matches with a distance not greater than 2.

Reference: TGGTACTTCTCCTACCCCCCA

Read #1: TACTT

Read #1

Reference: TGGTACTTCTCCTACCCCCCA
TACTT

Reference: TGGTACTT-CTCCTACCCCCCA
TACTT

Reference: TGGTACTTCTCCTACCCCCCA
TACTT
Exercise 2.2 c)

c) Solve the following read mapping problem instances. The distance function is the edit distance. All reads are of good quality. Write down all matches with a distance not greater than 2.

Reference: TGGTACTTCTCCTACCCCCCA
Read #2: CTTTC

Read #2

Reference: TGGTACTTCTCCTACCCCCCA
Reference: TGGTACTTC---TCCTACCCCCCA
Reference: TGGTACTTCTCCTACCCCCCA
c) Solve the following read mapping problem instances. The distance function is the edit distance. All reads are of good quality. Write down all matches with a distance not greater than 2.

Reference: TGGTACTTCTCCTACCCCCCA
Read #3: TCCTC

Read #3

Reference: TGGTACTTC - TCCTACCCCCCA
c
TCCTC

Reference: TGGTACTTTCTCCTACCCCCCA
TCCTC
c) Solve the following read mapping problem instances. The distance function is the edit distance. All reads are of good quality. Write down all matches with a distance not greater than 2.

Reference: TGGTACTTCTCCTACCCCCCA
Read #4: CCGCC

**Read #4**

Reference: TGGTACTTCTCCTACCCCCCA

Reference: TGGTACTTCTCCTACCCCCCA

Reference: TGGTACTTCTCCTACCCCCCA
Exercise 2.2 d)

d) This is optional. Think about how a simple read mapper could be implemented, for instance, by reusing the result of 1b). The input to the function could be a reference sequence, a read, a constant k and a distance function. The output should contain locations of the reference sequence where the read matches with a distance not greater than k. There is no need to take efficiency considerations into account. Implement it as a testable function/method in one of your favourite programming languages. Show the correctness of your implementation by comparing it with the result of c).
#!/usr/bin/env python

'''Primitive read mapper.

The program gets the reference and a read as the argument. It will print a
result TSV table.

USAGE: read_mapper.py REF READ

For example:  
'''

import sys
def begin_search(ref, read, k):
    previous_col = range(len(read) + 1)  # no free begin gaps
    # Store best match position for reverse search.
    best = None
    for i, c1 in enumerate(ref):
        current_col = [i + 1]  # no free end gaps
        for j, c2 in enumerate(read):
            hor = previous_col[j + 1] + 1        # horizontal in matrix
            vert = current_col[j] + 1            # vertical in matrix
            diag = previous_col[j] + (c1 != c2)  # diagonal in matrix
            current_col.append(min(vert, hor, diag))
        if current_col[-1] <= k:
            if best is None or current_col[-1] < best[1]:
                best = (i + 1, current_col[-1])
        previous_col = current_col
        assert best is not None
    return best
def edit_distance_search(ref, read, k):
    previous_col = range(len(read) + 1)  # no free begin gaps
    for i, c1 in enumerate(ref):
        current_col = [0]  # free begin gaps
        for j, c2 in enumerate(read):
            hor = previous_col[j + 1] + 1        # horizontal in matrix
            vert = current_col[j] + 1            # vertical in matrix
            diag = previous_col[j] + (c1 != c2)  # diagonal in matrix
            current_col.append(min(vert, hor, diag))
        if current_col[-1] <= k:
            ref_rev, read_rev = ref[:i + 1][::-1], read[::-1]
            ref_rev = ref_rev[:len(read) + k]  # no need to search more
            pos, score = begin_search(ref_rev, read_rev, k)
            yield {'begin': i + 1 - pos, 'end': i + 1,
                   'ref': ref[i + 1 - pos:i + 1], 'read': read,
                   'score': score}
    previous_col = current_col
if __name__ == '__main__':
    # Program entry point.
    if len(sys.argv) != 3:
        print 'Invalid number of arguments'
        print ''
        print 'Usage: read_mapper.py REF READ'
        print ''
        print 'Example: read_mapper.py TGGTACTTCTCCTACCCCCCA TACTT'

    ref = sys.argv[1]
    read = sys.argv[2]

    print 'BEGIN\tEND\tREF\tREAD\tSCORE'
    for match in edit_distance_search(ref, read, 2):
        print '%(begin)d\t%(end)d\t%(ref)s\t%(read)s\t%(score)d' % match
Exercise 2.2 d)

The program finds surprisingly many matches for the first pair:

```
17:01:00 tmp $ python edit_distance2.py TGGTACTTCTCCTACCCCCCA TACTT
BEGIN     END     REF      READ     SCORE
 3         6       TAC      TACTT    2
 3         7       TACT     TACTT    1
 3         8       TACTT    TACTT    0
 3         9       TACTTC   TACTT    1
 7         10      TCT      TACTT    2
 7         11      TCTC     TACTT    2
 9         13      TCCT     TACTT    2
 9         14      TCCTA    TACTT    2
12         15      TAC      TACTT    2
12         16      TACC     TACTT    2
12         17      TACCC    TACTT    2
```
Exercise 2.2 e)

e) To make yourself familiar with various read mappers, you should reproduce the Rabema benchmark that was partly introduced in the lecture. […]

Preliminaries

- Download SeqAn through SVN and compile razers3 and Rabema.
- Download and install samtools.
- Download and install BWA.
- Download and install Bowtie2.
- Download and extract rabema-data.tar.gz from Rabema homepage
Exercise 2.2 e)

Use RazerS 3 to build gold standard SAM file.
Exercise 2.2 e)

Prepare RazerS 3 “golden” SAM file for input to rabema_build_gold_standard.
Exercise 2.2 e)

Build gold standard intervals (GSI) file with rabema_build_gold_standard.

```
File   Edit  View  Search  Terminal  Help

Process genome seq #12[fwd]........1M
Process genome seq #12[rev]........1M
Process genome seq #13[fwd].......
Process genome seq #13[rev].......
Process genome seq #14[fwd]........1M.
Process genome seq #14[rev]........1M.
Process genome seq #15[fwd]........1M
Process genome seq #15[rev]........1M
Process genome seq #16[fwd]
Process genome seq #16[rev]Thread #0
  Masking duplicates took 0.00331764 seconds
  Compacting matches took 6.90451e-310 seconds
  Time for filtration 1.39644 seconds
  Time for verifications 5.17978 seconds
  Time for copying back 0.00247702 seconds
Finding reads took 6.74601 seconds

__FILTRATION_STATS__
Filtration counter: 5215634
Successful verifications: 25112
Dumping results took 2.88325 seconds
holtgrew@mouse ~/Development/seqan-trunk-build/release
16:03:04 release $ ./bin/rabema_prepare_sam -i out_gold.sam -o out_gold.prep.sam
holtgrew@mouse ~/Development/seqan-trunk-build/release
16:03:31 release $ samtools view -Sb out_gold.prep.sam >out_gold.bam
[samopen] SAM header is present: 17 sequences.
holtgrew@mouse ~/Development/seqan-trunk-build/release
16:03:37 release $ samtools sort out_gold.bam out_gold.by_coord
holtgrew@mouse ~/Development/seqan-trunk-build/release
16:03:41 release $
```
Exercise 2.2 e)

Run RazerS 3 in lossy mode and evaluate results using rabema_evaluate.
Exercise 2.2 e)

Run BWA and evaluate results using rabema_evaluate.