# Invitae Exercise B, Sajani Swamy

## Usage

python translate\_coordinates.py

## --genome-mapping-file GENOME\_MAPPING\_FILE

## REQUIRED. File specifying mappings (inputfile1.txt in exercise

## specifications)

## --transcript-processing-file TRANSCRIPT\_PROCESSING\_FILE, REQUIRED

## File specifying transcripts to process (inputfile2.txt

## in exercise specifications)

## --output\_file OUTPUT\_FILE, OPTIONAL

## Name of output file to write results to. Default is

## output.txt if none provided.

## Features Implemented

The current implementation enables the following features:

* Basic requirements of the coding exercise
* Mapping coordinates: genomic->transcript, transcript->genome
* Can process transcript alignments on the reverse strand (3’->5’)

## Input file format

**Genome-mapping-file**

* Same as input file 1 in the pdf exercise description
  + Chromosome name, 0-based starting position on the chromosome, and CIGAR string indicating the mapping
* We have an optional fifth column which indicates the direction of the mapping
  + “+” for 5’->3’
  + “-“ for 3’->5’
  + if this value is not provided a default of 5’->3’ is assumed

**transcript-processing-file**

* same as input file 2 in the pdf exercise description: transcript name, second column is a 0-based transcript coordinate
* We have an optional third column which indicates if the mapping is from genome >transcript or transcript->genome
  + “GENOMIC” for genomic coordinate->transcript coordinate
  + “TRANSCRIPT” for transcript coordinate->genomic coordinate
  + if the value is not provided, the default of TRANSCRIPT is assumed

## Output file format

* if this argument is not provided, the output file will be output.txt
* format is mostly similar to that described in the pdf but with some modifications:
  + col1: transcript name
  + col2: genomic coordinate
  + col3: chromosome
  + col4: transcript coordinate
* To accommodate the convention I adopted for mapping indels, coordinate which are in an insertion are output as A-B where A is the position immediately before the indel and B is the indel immediately after the indel.
* If there was an error in the input or otherwise during processing (i.e. the translation requested in transcript-processing-file was not possible), the requested translation coordinate is ‘ERROR’ (there is probably a better solution for this..).

## Unit Tests

Unit tests on the method which performs coordinate translation are found in tests/test\_class.py. There are several flavors of tests here. To add a new test, create a Mappings object (see examples in setup classes), and you can run tests in the ‘test’ method.

To run the tests, please enter:

nosetests test/test\_class.py

## High level architecture description

Main implemented in translate\_coordinate.py (translate\_coordinates method). This class reads in the files, does some error checking and performs the requested coordinate translations.

Description of classes :

**SequenceRange**;

* One object for every cigar operation and input sequence. Represents a contiguous interval of bases, in either transcript or reference.
* Holds information about start/stop positions of the interval and the cigar operation that generated the interval.

**Mappings class**:

* Main class which stores information about the alignment of the transcript on the genome and coordinates after each cigar operation
* Has 2 arrays of SequenceRange objects – one for transcript (referred to as ‘query’) and one for reference.
* When the Mappings object is initialized, the input cigar string is parsed and the sequence range arrays populated. After initialization, we should have query and reference arrays of length n, where n is the number of overall cigar processes.
* get\_pos is a generic method in this class which takes 2 lists of SequenceRange objects and can translate an input coordinate between the lists.

**CigarOperation**

* Simple class to store the operation type and length of any given cigar operation.

**GenomicMapping**

* Stores the information from the input file describing the transcript alignment to the genome (input file1 )
* Does some minimal error checking

## Assumptions about input

* Only cigar operations allowed are: MID (in theory N and X= should be OK, but I didn’t have time to test them)
* Since cigar strings in SAM format represent alignment of an input sequence to the forward genomic strand, we will assume that any reference position is with respect to the forward strand.

## Translating Coordinates

When we translate a coordinate from genome->transcript or vice versa, we call a generic function which takes in 2 sequence ranges (SR1 and SR2), a mapping position relative to SR1 that we want in SR2 and 2 Booleans indicating the mapping orientation of SR1 and SR2. If SR1 and SR2 are both in the reverse direction we throw an error since at least one of the ranges should represent the forward strand reference since the cigar string by definition is specified only for the forward strand reference.

The first step is to locate the range in which the input coordinate is found in SR1. We need to account for both forward and reverse strand mapping when doing this indexing. It’s assumed that the number of cigar operations for any mappings in the input file is modest, and as such linear traversal of the list is reasonable. Once the index of the range holding the input coordinate is found in SR2, we simply look up the range with the same index in SR2. We find the offset of the coordinate in range based on start/stop positions and return this position.

If the base that we wish to translate is found in an inserted sequence, the program returns the base immediately prior and immediately after the insertion. For example, in the tables above, for the 2I operation, transcript coordinate 8 in the 5’->3’ mapping doesn’t have a single, direct mapping in the reference. There are several conventions that can be applied, but I chose to return the coordinate immediately before and after the insertion in a tuple. So, the output for transcript coordinate 8 would be (10,11)

## Examples:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **5’=>3’ Mapping** |  |  |  |  |  |
| Cigar op | 8M | 2I | 5M | 4D | 3M |
| Transcript coordinates (start,stop) | 0 7 | 8 9 | 10 14 | 14 14 | 15 17 |
| Reference coordinates(start,stop) | 3 10 | 10 10 | 11 15 | 16 19 | 20 22 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **3’=>5’ Mapping** |  |  |  |  |  |
| Cigar op | 8M | 2I | 5M | 4D | 3M |
| Transcript coordinates (start,stop) | 10 17 | 8 9 | 3 7 | 2 2 | 0 2 |
| Reference coordinates(start,stop) | 3 10 | 10 10 | 11 15 | 16 19 | 20 22 |

## Strengths/Weaknesses of implementation

**Strengths**

* Straightforward implementation
* Unit test which covered main translation feature pass – correct implementation
* Modular
* Code is somewhat flexible and extending code for remaining ‘bells&whistles’ requirements should be possible
* Much of code is documented

**Weaknesses**:

* If we have a lot of queries to process or the cigar strings/mappings are very long, some parts of the code could be inefficient possibly (need to benchmark). One place to start could be the get\_pos method.
* Need to refactor some of the main methods (especially translate\_coordinates and get\_pos) – they are too long
* Need more thorough error checking of the input
* Variable names could be better
* Might have missed some more corner cases for testing

## Bells & Whistles Question 4

*A real-world implementation of this code would need transcripts from external sources. Discuss where and how you might obtain these?*

Answer probably varies depending on the application and goals of the analysis! Some sources of transcripts:

* Databases: I would almost certainly start with RefSeq transcripts (download from NCBI, ftp://ftp.ncbi.nlm.nih.gov/refseq/), which is generally a conservative list but high quality. Alternately other sources like Encode or Ensemble are possible