Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computational Biomedicine I
Fall Semester 2021

Project 1: Efficient Search and Read Alignment

Assigned on: **21.9.2021** Due by: **11:59pm on 04.11.2021**

Overview

With this exercise sheet, we will present the first practical project of the class. The topic of this project is the alignment of high-throughput sequencing read data to a reference genome sequence. The goal is to build a simple alignment software tool that is capable of generating alignments for the given input data and outputing them in a given reference format.

In particular, in this exercise you will:

a) decide on and implement a genome indexing strategy

b) decide on and implement a read alignment method

We have split the work into a few main work packages, outlined below.

Preliminaries – Input Data

We have generated two kinds of input data sets. The first kind (`output_tiny_30xCov*`) is a very small set of approx. 1100 reads generated from a short, artificial reference sequence of only 5000 bases. Along with the read data and the reference sequence, this set also contains the optimal alignment output, as was generated along with the sequence. The alignment output is provided in SAM format. Although the aligner you design does not need to output in this format, it is provided as ground truth to assist you during development. You can use this data set as a quick-to-run test for implementation correctness.

The data sets of the second kind (`output_MxCov*`) consist of read sets simulated using different coverage values from chromosome 22 of the human reference genome. These sets are much more realistic, but still quite small compared to typical sets from whole genome sequencing. We provide three different data sets, for coverage values `M` of 5, 10 and 30, respectively. These data sets will be the input to your program. You can work with the smallest coverage for the project and use the higher coverages for benchmarking at the end.

For each data set, we provide the genome sequence the reads were sampled from and should be aligned to, as well as two read files in fastq format. The two fastq files contain read pairs, where one read originates from one end of a DNA fragment and the other read from the other end. The order of reads in the two fastq files (`*[12].fq.gz`) is the same.

All input data is available for download from:

▷ https://public.bmi.inf.ethz.ch/eth_intern/teaching/cbm_2020/cbm_2020_project1/

Please download all data and familiarize yourself with the provided data formats. In case of any questions, please use moodle. All data is text-based and should be human-readable.

Preliminaries – Code Structure

To assist in structuring your code, we have some basic classes in `aligner.py`. This includes the `Seed` and `Alignment` storage classes, helper methods for reading FASTA and FASTQ files, and the abstract `ReferenceIndex` and `Extender` classes from which your methods will inherit (more details on these in the sections below).

As an example, we have provided a naïve implementation of these two classes in `dummy_aligner.py`. This dummy implementation aims to help you understand the use of different classes.

We have also provided an IPython notebook for evaluating the accuracy of your aligner. You may update the last two lines of the notebook to reflect the names of your implemented index and extender classes, respectively.

When developing the libraries below, we emphasize that the index data structure(s) and search algorithm must be implemented by you. Aside from libraries which provide for low-level vectors or vector operations (such as numpy or Numba), you are not allowed to import libraries to provide functionality. In case of any doubts or to ask whether a particular library would be allowed, please ask your TA team on Moodle.

Work Package 1.1 – Genome Index

After learning about different indexing strategies for sets of strings, you should discuss and decide on a strategy you would like to pursue for the projects. Aspects you should include into your discussion are: time and effort to implement said indexing, space and time requirements to construct the index as well as space and time requirements for the alignment task.

The goal of this work package is to devise and implement a method that generates an index on the input data. You must provide a class which inherits from the `ReferenceIndex` abstract class and implements its methods. Alongside indexing the reference, it must also be able to return a set of alignment seeds from which to begin alignment given a query sequence. Ideally, this index can be stored on disk and re-used at a later time. This way the up-front cost for index-creation has to be paid only once.

Work Package 1.2 – Sequence Alignment

Using the genome index generated in work package 1.1, you should design and implement a read alignment strategy.

Think about and discuss which cost functions are useful and appropriate from a biological point of view.

The goal of this work package is to develop an alignment software tool that can find a high-scoring position of one or many reads provided in FASTQ format in a fully indexed genome. You must provide a class which inherits from the `Extender` abstract class and implements its methods.

Submission

For development and versioning of your code, we provide each group with a git repository using the department's GitLab instance. The same git repository will also be used for your project submission. Once the project is due, we will take the code on the `master` branch of your group's repository and execute it on the input data.

Code needs to be executable in the provided conda environment. If additional packages have been approved for use by the TA team, please provide a conda recipe. Any solution that is not using Python needs to be provided with Python wrappers which will allow it to be used in the manner described above. Please note that submissions written in languages which produce faster code (such as C/C++) will have their run times scaled to be better comparable to Python submissions. **Please understand that for practical purposes any code that a qualified TA is unable to get running within 10-15 minutes due to missing information, cannot be graded**.

Each Gitlab submission needs to be accompanied by the recording of a 5 minutes presentation per group member. The video upload form can be found below:

▷ https://forms.gle/sTtx8rUgEeeboT6d7

Each group member should give a brief summary of the group's solution and briefly outline the parts they worked on themselves. Everybody should be using their own slide deck (or other preferred media), but exchanging within and across groups is encouraged. A platform for discussion across groups is provided via moodle.

So one more time: If there is any additional knowledge required to run your code or necessary prerequisites to be made, please add a README file containing all relevant information to your repository. If any part of your code requires compilation, you are required to provide a Makefile.

Grading

In order to pass, you need to meet the benchmark defined in the grading scheme. The grading scheme for each task is defined as follows:

- For Work Package 1.1, the upper bound for run time is 15 minutes.
- For Work Package 1.2, the average rate of alignment is 1000-5000 reads per minute (in python) and 40000 reads per minute (in C++). The mean edit distance between the aligned and true sequence should be less than 0.1.

In order to pass with honors, the performance of a group must be exceptional (with respect to your peers) in terms of the run time. Additionally, the mean edit distance between the aligned and true sequence should be less than 0.01.