Department of Computing and Information Systems

COMP 90016

Assignment 1

Release date: 20th of March 2018

Due date: 9th of April 2018 (11:59pm)

This assignment is worth 10 marks, or 10% of your final score

In this assignment we are deepening our understanding of alignment algorithms once more. Is it getting old yet?

The Hamming distance aware aligner from Workshop 4 helped us to map reads that are not identical to the reference. However, for a high number of reads and a long reference genome, this method quickly reaches its limits. Specifically, the complexity of the Hamming distance aligner is $O(ngl)$, where *n* is the number of reads, *g* the length of the genome, and *l* the length of the reads.

The task of this assignment is to improve this complexity significantly.

Tasks:

1. Implement an alignment algorithm that makes use of an indexing strategy of the reference genome.
   This task is to be carried out in two steps:
   I. Write a program *indexer.py* that takes as input a FASTA reference file and an integer value k (on the command line). The program is to extract every k-mer from the forward strand of the reference and store its position in a dictionary. Specifically, the keys to the dictionary are k-mers and the values are lists of positions at which this k-mer occurs (1-based coordinates).
   After completing the dictionary, write the index to a file (*index.txt*). The first line of the index should be starting with the word "INDEX:" followed by the name of the reference genome (as stated in the reference's FASTA file) and the value k. Following this, the entire dictionary should be printed, one line for each key value pair.
   For example, the start of an index file could look like this:
   INDEX: toy_reference 4
   AAAAA 5 19 27
   AAAAC 6 28 39
   AAAAG 20
   AAACA 7 40
   …
   II. Write an alignment algorithm *aligner.py* that utilizes the pre-computed information from Task 1.1: The (command line) inputs to the program should be a reference index (see above), a reference file, and a FASTQ read file.
   The aligner should first parse the index file to re-create the dictionary of k-mer position lists. From the first line, the reference name and value for k are parsed and saved.
   To align the reads, the program should make use of the index. Identify the best alignment possible with Hamming distance 2 or lower for the read and its reverse complement. Alignments should be reported as before (Workshop 4, group assignment): one line per read, left-most position in the reference genome, fields *READ_NAME, REF_NAME, POS, STRAND, NUMBER_OF_ALIGNMENTS, HAMMING_DISTANCE.*
   If you are struggling to conceptualize an algorithm by yourself, these guidelines might be useful to you: For each read, extract the k-mers, and for each k-mer query its existence in the reference via the index. Investigate all positions for a k-mer. Identify

the section of the reference that matches up with the read (due to sharing the same k-mer). Then, compare all other bases of the read and reference around this k-mer. If an alignment of Hamming distance 2 or lower is possible, save the alignment, unless a better one was already found. Repeat for the reverse complement of the read. To avoid needless computation, the aligner should not investigate the same alignment twice (that is, the same read and the same reference position).

Task 1 is worth 6 marks. Marks will be deducted for inaccurate mapping positions, duplicate investigation of alignments, incorrect output format (both index and alignments), amongst other criteria.

2. Investigate the runtime of your new aligner. Argue theoretically, the complexity of your program. For this purpose, you can assume constant time to query the index. Then compare the alignment time (not including index creation) of your program and the program from Workshop 4 in practice. Do so by running your aligner on the three different reference files provided (*ref1.fa, ref2.fa, ref3.fa*), and k=13. Measure the runtime with the Unix *time* command. Running the program from Workshop 4 takes a considerable time, so use the following measurements for your visualization: ref1:16m, ref2:32m, ref3:50m. Plot the runtimes for both aligners and argue whether the measured values compare with your expectations of theoretical reasoning.

Task 2 is worth 2 marks. Marks can be deducted for missing theoretical considerations, missing or uninformative visualization of practical results, and missing or minimal discussion.

3. Extend your algorithm to further investigate mismatches to the reference. Submit the modifications as a separate program (*base_qualities.py*). During the alignment stage, keep track of each mismatch position (within the read) of the best alignment (on a tie, only the one reported) and their quality values. At the end of the alignment stage output or plot this information and compare: a) the distribution of base quality scores of all the reads for each position (see Workshop 2); b) the distributions of quality scores of all mismatches (also by position). Discuss the similarities and differences in the two distributions. Expected are about no more than 1 page of discussion, including the plot, information about the differences and similarities of the distributions, your thoughts and insights on the differences' cause, and the utility of this information.

Task 3 is worth 2 marks. Marks can be deducted for an incorrect program, missing or uninformative plots, and a missing or minimal discussion.

Notes:

- If you cannot develop an alignment algorithm for Task 1, you can commence with the previously developed Hamming distance aligner for Task 3.
- To visualize the results in Task 3, the boxplot used in the workshop may be a good choice, but you may also consider other options.
- The format of the text file in Task1.1 (other than the first line) is up to you, but you have to make sure to keep it consistent with the next step.
- As a quality check, your aligner should produce identical results to the Hamming distance solution from Workshop 4 (but faster). Note though, that this is guaranteed, as the seeded alignment technique is a heuristic, unlike the exhaustive search from Workshop 4.
- Use the *user* field from the output of the *time* command for your measurement in Task 3.
- Note that the *REF_NAME* field in the alignments is largely meaningless for this assignment. We keep it around to mimic the SAM format a bit more closely.