CMSC701 – Class project

Due: December 2, 2013 at midnight

You will have the choice of two projects (A, and B, described below) and can work in teams of up to 2 people. The grade will be the same for both team members (i.e., I won't adjudicate disagreements between team members).

Deliverables:

1. Source code (documented) and instructions on how to install and run the code on a Linux system. You can use any programming language. If you use a non-standard library (i.e., something a typical Linux system does not have installed) please either include it into your package, or provide instructions on how to install/use it. If necessary, provide hard-linked binaries in addition to the source code.

2. Document describing your algorithm and results of experiments (e.g., how you convinced yourself that it is linear time). Please also include some results of comparisons to open-source software you could use to solve this problem.

Grading:

Total: 100 points (broken up as follows)

Code is correct and executes: 60 points
Code is well documented (including comments, and documentation on how to install and run) : 20 points
Document describing algorithm is well written and complete: 20 points

Bonus points:
Fastest program (awarded separately for project A and B): 10 points
Lowest memory usage (awarded separately for project A and B): 10 points

To qualify for the memory usage award your runtime cannot be in the bottom 5 within the class (i.e., you can't simply create a brute-force algorithm).

And now for the description of the projects. Test data are available at
ftp://ftp.cbcb.umd.edu/pub/data/CMSC701

Project A. Mapping short sequences (sequencing reads) to a collection of genomes

For this project you will be given a collection of short reads (in fastq format) and a database comprising multiple genome sequences (in fasta format). The output of your program will consist in a TAB-delimited file comprising the following information:

read ID, genome ID, genome coordinate, # of edits

Only reads that map to a genome (within a user-specified number of insertions/deletions/mismatches) need to be reported. For every read you only need to report the best match, but you need to report all matches that have the same best score, whether against the same genome or multiple genomes. For example, assume read 1 matches genome A with 3 mismatches at position 1000000, genome B with 2 insertions at position 15000, genome B with one insertion and one mismatch at position 50000, and genome C with 2 mismatches at

position 1000, the output should be:

1 B 15000 2
1 B 50000 2
1 C 1000 2

This project is inspired by the paper:
Schneeberger, K., J. Hagmann, S. Ossowski, N. Warthmann, S. Gesing, O. Kohlbacher and D. Weigel (2009). "Simultaneous alignment of short reads against multiple genomes." Genome Biol **10**(9): R98. http://genomebiology.com/2009/10/9/R98

Project B. Finding all reads that match a reference read set.
For this project you will be given two collections of reads (in fasta format). One of these will be smaller and will be termed the 'reference', while the second will be termed the 'query'. Your task is to find, for each read in the reference, the set of reads in the query that match it within a specific edit-distance threshold (also given in terms of number of insertions/deletions/substitutions). You can assume that all reads are expected to start at the same position, i.e., your alignment algorithm should penalize insertions and deletions at the beginning of the sequence, but must tolerate insertions and deletions at the end of the sequence. For example, the following alignment should be considered as 'perfect':

```
AGGATCATGCATTTGA
AGGATCAT--------
```

while the alignment:
```
AGGATCATGCATTTGA
-GGATCAT--------
```

represents an edit distance of 1.

The output of your program should comprise a series of lines, each of which starts with the identifier of a reference sequence followed by a space-separated list of all the query sequences that match it to within the user-specified edit distance cutoff.

This project is inspired by the paper:

Ghodsi, M., B. Liu and M. Pop (2011). "DNACLUST: accurate and efficient clustering of phylogenetic marker genes." BMC Bioinformatics **12**: 271. http://www.biomedcentral.com/1471-2105/12/271

Useful resources

SeqAn library – http://www.seqan.de – C++ library for processing of DNA sequences (includes suffix arrays, BWT, etc.)

Gsuffix – http://gsuffix.sourceforge.net – A C library implementing various suffix tree and array algorithms

FMindex – http://www.di.unipi.it/~ferragin/Libraries/fmindexV2/ - the original FM index implementation of Ferragina and Manzini

Python suffix tree library - https://github.com/JDonner/SuffixTree

Python suffix array library – https://code.google.com/p/pysuffix/

Also look at Bio::Perl, BioJava, BioPython – libraries that include many utilities for processing and even aligning biological objects.