CAP5510 - Bioinformatics Fall 2018, Homework 1

Due date: 10 / 25 / 2019
Turn in hard copy of report in class.
Upload source code and Readme file using e-learning.

September 30, 2019

This is a programming assignment. The purpose is to gain experience on sequence alignment. You can work in teams of two (or do it alone if you prefer). In this homework, you will implement

1. Simulator for sequence generator
2. Simulator for sequence partitioning
3. Sequence assembler

The algorithms you need are available in my course slides at `www.cise.ufl.edu/~tamer/teaching/fall2019/lectures` (Slides 2). Below is a description of each of the three items above.

Simulator for sequence generator. This program will generate a set of similar sequencs as follows.
INPUT:

1. integer: n = sequence length
2. four integers: a c g t = fraction of letters A, C, G, and T
3. integer: k = number of sequences
4. real number: p mutation probability in [0:1] interval
5. string: output file name

OUTPUT:The program will output k nucleotide sequence in Fasta format. The first sequence will have n letters. It will be randomly generated. Define sum $s = a + c + g + t$. You will generate each letter A, C, G, T with probability $\frac{a}{s}$, $\frac{c}{s}$, $\frac{g}{s}$, $\frac{t}{s}$, respectively. The next $k - 1$ sequences will be a copy of the first sequence, but each letter will be mutated with replace or delete with probability p. The output file will contain all k sequences in Fasta format.

Simulator for sequence partitioning. This program will partition the sequences in a file into small fragments as follows.

INPUT:

1. string: input file name
2. integer: x = minimum fragment length
3. integer: y = maximum fragment length ($y \geq x$)
4. string: output file name

OUTPUT:The program will read the sequences in the input file, which is in Fasta format. This input file is the output of the first program above. It will then partition each sequence into smaller fragments. The length of each fragment is a random number in the range $[x : y]$. As you chop the sequences into small fragments, if there is a small piece left with length less than x, throw away that fragment. The output file will contain all fragments in Fasta format.

Sequence assembler. This program will assemble the fragments of sequences in a file into one long sequence as follows.

INPUT:

1. string: input file name
2. integer: s = score for match (positive integer)
3. integer: r = penalty for replace (negative integer)
4. integer: d = penalty for delete/insert (negative integer)
5. string: output file name

OUTPUT:The program will read the sequences in the input file, which is in Fasta format. This input file is the output of the second program above. It will then combine fragments using a greedy strategy as follows.

1. Align all fragments f_i and f_j with each other using dovetail alignment and compute the alignment score $v_{i,j}$
2. **Repeat**
   (a) Merge the two fragments f_i and f_j with the largest alignment score $v_{i,j}$ into a new fragment f', if $v_{i,j} > 0$
   (b) Replace f_i and f_j with the new fragment f' and compute align f' with all the other fragments in the current set using dovetail alignment.
3. **Until** one fragment is left in the set or the largest alignment score is negative
4. Write the longest fragment you assembled into output file in Fasta format.

Return. You will return your source code, executable, and a Readme file in a single tar-zip file through Canvas. One submission per team is enough.

Final details.

- You can use any programming language. Make sure that the program runs on CISE linux machine (such as thunder or storm).
- All three programs should be your own implementation.
- Use the following names for the three programs above *hw1-1*, *hw1-2*, *hw1-3*, as the executable of your code. For example, a command line to partition a file (second program above) should look like

  `> hw1-2 inputfile 100 150 outputfile`

  to partition the input into fragments of length 100 to 150.

Here are some sample parameters you can use to test your code:

```
> hw1-1 10000 25 25 25 25 10 0.005 outputfile
> hw1-2 inputfile 100 150 outputfile
> hw1-3 inputfile 1 -1 -3 outputfile
```