Assignment 8 Restriction Enzyme Simulation

1 Overview

In this assignment your program will process a DNA string. Here, the program will be given two arguments on the command line. One will be the name of a file containing lines that identify ***restriction enzymes***, and the second will be a file containing a DNA string. Let's call the first file the *enzyme file* and the second, the *dna file*. For every line in the enzyme file, it will apply the information in that line to the DNA contained in the dna file. Roughly put, a restriction enzyme is a pattern that acts like a pair of scissors that cuts DNA according to the pattern. It specifies a place in the DNA string at which to cut the DNA. This is explained in more detail below.

The details of the assignment are specified in the *Detailed Requirements* section below. The *Background* section that follows is a short tutorial on the subject of restriction enzymes. It includes some background from previous assignments so that this assignment is self-contained. This assignment will give you experience in reading from files and in more sophisticated pattern-matching.

2 Background

2.1 DNA and Nucleotides

A *DNA string*, also called a *DNA strand,* is a finite sequence consisting of the four letters `A`, `C`, `G`, and `T` in any order[1]. The four letters stand for the four *nucleotides*: *adenine*, *cytosine*, *guanine*, and *thymine*. In this assignment, the nucleotides will always be in *uppercase* format. Nucleotides, which are the molecular units from which DNA and RNA are composed, are also called *bases*. Each nucleotide has a *complement* among the four: `A` and `T` are complements, and `C` and `G` are complements. Complements are chemically-related in that when they are close to each other, they form hydrogen bonds between them.

In its most common form, DNA is actually a double helix consisting of two strands that wrap around each other. Each DNA strand has direction. Direction is usually indicated by putting the symbol 5' at one end and the symbol 3' at the other. The 5' and 3' refer to the names of the carbon atoms to which these ends attach. For example,

```
5'-GTATCC-3'
```

is a fragment of DNA that runs from the 5' to the 3' position. The 5' end is called the *upstream* end, and the 3' end is the *downstream* end. The two strands of nucleotides in DNA are in reverse directions of each other. In other words, if you could unwind the helix so that the two strands were lying on a flat surface parallel to each other, in one strand the 5' end would be to the left, and in the other, it would be to the right. The two strands are chemically-related because the bases that would be across from each other in the strands lying on the table are complements of each other. For example, the two strands below

```
5'-G T A T C C A A T G C C-3'
   | | | | | | | | | | | |
3'-C A T A G G T T A C G G-5'
```

could be a fragment of the unwound double helix. The vertical lines connect complements in the forward and reverse strands to each other. Each `C` in one is matched by a `G` in the other, and each `A` is matched by a `T` in the other.

[1]It does not matter whether they are in uppercase or lowercase.

2.2 Restriction Enzymes

Bacteria produce special enzymes that can cut their DNA at specified sites, called *cleavage sites.* The cleavage site is a position between two nucleotides in the DNA. The enzyme finds its site by a type of biological pattern-matching. The pattern specifies where in the DNA the enzyme will match. The place on the DNA molecule that matches the pattern is called the *recognition site.* For example, the enzyme *EcoRI* has a recognition site defined by

```
5'-G'AATTC-3'
```

The apostrophe `'` between the `G` and the `A` is the cleavage site. This means that *EcoRI* will search for a substring of the DNA consisting of the bases `GAATTC` in the 5' to 3' direction, and if it finds a substring of the DNA that matches `GAATTC`, it will cut the DNA between the `G` and the first `A`. So, if the DNA string is as below (with the parts that match the pattern, i.e., the recognition sites, in ***bold italics***)

ATGAAAGGGTTTCCCTTT***GAATTC***CCCATGGTATTGTTGCCG***GAATTC***TTTCCGGCCCCC

it will be cut into the three pieces

```
ATGAAAGGGTTTCCCTTTG    AATTCCCCATGGTATTGTTGCCGG    AATTCTTTCCGGCCCCC
```

by *EcoRI.* The restriction enzyme *NotI* is defined by

```
5'-GC'GGCCGC-3'
```

which indicates that it will find all occurrences of the string `GCGGCCGC` in the 5' to 3' direction and cut the DNA between the first `C` and the second `G`.

You may have noticed that if you form the complement of `GAATTC`, you get `CTTAAG`, which is the string `GAATTC` spelled backwards. Similarly, the complement of `GCGGCCGC` is `CGCCGGCG`, which is the string `GCGGCCGC` spelled backwards. Certain types of restriction enzymes have this property, which we call *palindromic.*

Some restriction enzymes have a cleavage site outside of the recognition site. *AceIII* is defined by

```
CAGCTCNNNNNNN'
```

The `N` is a symbol that matches any of `A`, `C`, `G`, or `T`. It is like writing a regular expression

```
CAGCTC.......'
```

Therefore, this enzyme cuts the DNA between the 7th and 8th nucleotides after its recognition site, because there are 7 `N`'s after the `C`. For example, the short fragment of DNA

```
CAGCTCAAATGCCAGGGGGGG
```

will be cut between the 5th `C` and the `A` because there are 7 characters in `AAATGCC:`

```
CAGCTCAAATGCC  AGGGGGGG
```

In actuality, many of these restriction enzymes cleave both strands of the DNA at once, and not necessarily at the same position. We are going to simplify the problem in this assignment and assume that the DNA is single-stranded and that it is cut only as described above.

3 Detailed Requirements

The program must be run with *two* command line arguments. Suppose the program is named `cleave.pl`. Then the command

```
cleave.pl enzyme_file mydna_file
```

will cause the program to read the `enzyme_file`, and for each enzyme in the file, to apply that enzyme's cleaving to the DNA in the file named `mydna_file`. Applying an enzyme to the DNA will result in a set of smaller fragments of DNA, which will be placed in an output file. There will be one fragment per line in the output file, arranged in the order in which they occur in the input file. For example, if the DNA is cut at positions 20, 175, 300, and 350, there will be five fragments, consisting of the DNA from 1 to 20, 21 to 175, 176 to 299, 300 to 350, and 351 to the end. Here the position is the base after which the cut occurs.

The name of the output file is constructed by appending the name of the restriction enzyme to the name of the DNA file, with an underscore in between them. For example, if the enzyme is *EcoRI* and the DNA file is named *BC161026*, the output file should be named *BC161026_EcoRI*. This implies that the number of output files is equal to the number of restriction enzymes in the `enzyme_file`.

The enzyme file will be a very simplified and modified version of a format known as the *Staden* file format. Each line in the file has the form

```
enzyme_acronym/recognition_sequence/
```

The cut point will be denoted by an apostrophe in the recognition sequence. An example of an input enzyme file is:

```
AatI/AGG'CCT/
AatII/GACGT'C/
AbsI/CC'TCGAGG/
AccII/CG'CG/
AccIII/T'CCGGA/
Acc16I/TGC'GCA/
```

The first line is the enzyme named *AatI* and its cut point is between the second `G` and the first `C`. Notice that none of the enzymes above have the `N` symbol. In an actual Staden file, there are many symbols besides `A`, `C`, `G`, and `T` in the recognition sequences.

To simplify this assignment, the only letters that you will find in the enzyme recognition sequences are A, C, G, and T. *But you should be aware that this is a very big simplification of what really happens.*

All of the DNA in the DNA file will be on a single line, possibly with a terminating newline character. It might be in upper or lowercase. Your program should work regardless of the case of the DNA bases.

4 Error Checking

If either file on the command line cannot be opened, the program should display a specific message related to the error and then exit. If the DNA file contains any symbols other than `A`, `C`, `G`,`T`, the program should exit with an appropriate error message. For simplicity, the program can assume that the enzyme file is in the proper format and contains no spaces. The program does not have to check that the enzyme file is in the wrong format.

5 Program Considerations

This section tells you the large chunks that you need to consider creating.

1. Check that the command line arguments are correct and valid and handle the errors.
2. Open the enzyme file, and from each line, extract the following information: the name of the enzyme, the pattern, and the distance from the start of the pattern to the cleavage site. How will you store this information? For each enzyme name there are two pieces of associated information: the pattern and the cleavage position.
3. Open and read the DNA file.
4. For each enzyme, create an output file with the proper name, and then apply the enzyme to the DNA, cutting it into pieces, and putting each piece on a separate line in the output file. Close the output file when you are finished with it!

The hardest part is cleaving the DNA. As a reminder we learned about Perl's special variables, $‘, $&, and $’. One or more of them might be useful. Equally important, you will need the `substr()` function. Read about it. Remember that after the DNA is cleaved, the enzyme will start looking for a match in the part immediately after the cleavage site. As an extreme example, suppose we had an enzyme with the recognition sequence `AA'AAA` and the DNA string was

```
AAAAAAAAAAAAAAA
```

Then it will be cleaved first into

```
AA AAAAAAAAAAAAA
```

and again into

```
AA AA AAAAAAAAAAA
```

and so on. The point is that after the enzyme is applied, the DNA string to be matched is the one starting right after the cleavage site.

Finally, as a reminder, the program must have proper documentation and comply with the Programming Rules.

The output can contain nothing but the DNA fragments – no messages to the user, no extra lines, no spaces. The output might be needed again as the input to a different program, and it is important that it is in the format that I described above.

6 Testing Your Program

All programs must be thoroughly tested before they are released to users. Create some sample input files of a small size and manually figure out what the outputs should be. Run your program and make sure that your output matches the one you manually computed. There are a few DNA files in the directory `/data/biocs/b/student.accounts/cs132/data/nucleotides`. Each is in three formats: a single line with no newlines, FASTA format, and a text format with newlines. More files can be created using the Perl script in the demos directory that makes random DNA strings. In addition, you can download DNA files from the GenBank or other such archives.

7 Rubric

This homework is graded on a 100 point scale. The program will be graded primarily on its correctness. This means that it does exactly what the assignment states it must do, in detail. Correctness is worth 70% of the grade. Then it is graded on its documentation and design. Comments and appropriate naming of variables are worth 20%; good design another 8%. Naming all files and directories correctly is 2%.

8 Submitting the Program

This assignment is due by the end of the day (i.e. 11:59PM, EST) on Thursday, November 29; it is okay to submit it by 6:00 AM of the following day.

Name your program `cleave.pl`. To submit your project, you must follow the instructions below exactly! Do not deviate from these instructions.

To be precise:

1. Login using `ssh` to `eniac.cs.hunter.cuny.edu` with your valid username and password, and then `ssh` into any `cslab` host. Do not forget this step. You will not be able to run the `submithwk` command on `eniac`.
2. If you did not do the work on one of the computers in our network, then upload the Perl program file into your home directory. Create a directory named `your-username-hwk8` and put the Perl program into it.
3. Run the command

   ```
   zip -r your-username-hwk8.zip your-username-hwk8
   ```

   This will create the file `your-username-hwk8.zip`. The `zip` command is a special command that compresses the files in the directory and creates a new file that can later be extracted by the `unzip` command. So it will create a "zip file" named `your-username-hwk8.zip` containing your `your-username-hwk8` directory and any files it contains. For example, I would run

   ```
   zip -r sweiss-hwk8.zip sweiss-hwk8
   ```

4. Run the command

   ```
   /data/biocs/b/student.accounts/cs132/bin/submithwk 8  your-username-hwk8.zip
   ```

   Do exactly this. Do not mistype it. The command will create a copy of the file `your-username-hwk6.zip` in the directory

   `/data/biocs/b/student.accounts/cs132/hwks/hwk8`

   It will be named `hwk8_`*`username`*, where *`username`* is your username on the network. You will not be able to read this file, nor will anyone else except for me. If you decide to make any changes and resubmit, just do all the steps again and it will replace the old file with the new one. I will be able to unzip the file, extracting whatever files you created. Do not try to put your file into this directory in any other way - you will be unable to do this.
5. **Do not put anything in this directory other than the file cleave.pl. You will lose one point for each file that is in this directory that does not belong there.**

Although these instructions may seem complicated, they simplify the way you submit your work and the way I can retrieve it. If you make mistakes, just start over. If things don't seem to work out, post a question on Piazza with the details included.