Spring 2012 CSC 448: Bioinformatics Algorithms Alexander Dekhtyar

Lab 3-1: Simple Contig Recovery (string overlap)

Due date: April 24.

About the Lab

This is a joint lab with **CHEM 441**. The **CSC 448** teams remain the same as in prior labs. The pairing with the **CHEM 441** students remains the same, unless Dr. Goodman explicitely makes reassignments.

The goal of this lab is to build software that assembles a number of separate DNA strings (contigs) into a single string, and verifies and updates the information about the coding regions in the new string to the new coordinates within the assembled string. The **CHEM 441** students have already started working on the requirements documentation for you, so it should be ready for the discussion by the time you meet in the lab.

Time	CSC 448	CHEM 441
April 17 lab	Studying assignment	Completing requirements
April 17 lab	Discuss assignment/map out solution	
April 17 – 19	Software development	Data collection
April 19 lab	Software development	
April 19	Prototype delivery/discussion/use	
April 19 lab	Software development	Software use
April 24 lab	Preparing deliverables and submission	

Lab Assignment

As in **Lab 2-2**, your assignment consists of two parts:

1. Development of the Boyer-Moore algorithm for string matching.
2. Use of the developed Boyer-Moore algorithm to prepare software for simple contig assembly [1] and verification/recomputation of the coding region information.

Both parts are briefly discussed below.

Boyer-Moore Algorithm

Implement the Boyer-Moore algorithm for string matching as discussed in class. Please note the following rules:

- Implement the algorithm and all its parts (bad character rule, good suffix rule, KMP's π function) from scratch.
- The inputs to the algorithm functions/methods shall be actual strings stored in a straightforward way (e.g., as a `String` variable in Java), plus whatever other parameters necessary.
- You are allowed to use the string length function/method in working with your string data.
- In all other cases, you must treat your string data as an array of individual characters. **No string manipulation functions/methods** (e.g., string comparison, string concatenation, search for a substring in a string) **are allowed**[2].
- Your implementation shall be alphabet-agnostic. That is, your code shall work properly with strings in any alphabet. Recall that bad character rule requires you to know the alphabet. Your code shall accommodate it properly in one of two ways:
  - Using as the alphabet the **effective alphabet** in which the two strings given to you are written. E.g., the actual alphabet for strings `"alpha"` and `"betalphalpha"` may be the 26-character Latin alphabet, but the effective alphabet you need to know is {`a,b,h,l,p,t`}.

[1] Simple contig assembly means that the order in which the contigs should be assembled is known to you in advance. A more general problem of contig assembly in the absense of the order information is NP-complete. Algorithms for it may be discussed closer to the end of the quarter.

[2] The reason is simple. We want you to implement string matching efficiently. Use of existing string functions/methods inside a loop is not an efficient way of solving the problem, because these functions themselves use standard string matching algorithms to do their work.

  - Passing the alphabet as the parameter to the Boyer-Moore algorithm implementation.

- Your program cannot limit the size of the string input. The actual inputs used in the second part of the assignment will be on the order of tens of thousands of characters. Your program needs to be able to handle strings up to hunderds of thousands of characters long. If traditional ways of managing strings that long in your programming language of choice fail (e.g., if this goes beyond the limitation of, say, Java's `String` data type) you must implement your own abstract data type for management of long strings. You can assume that all data given to your program will fit in main memory, but beyond that, the string sizes can grow large.

- Write a simple `main` program verifying the correctness of the work of your program.

Contig Assembly

As usual, each team will receive the specific instructions for this part of the assignment from their **CHEM 441** partners. The instructions in this document are just a brief general overview.

The goal of the **CHEM 441** teams on this stage is to assemble the genome fragment they are studying. The data they need to study is currently broken into a number of individual "chunks" called *contigs*. When combined, the contigs should form a single DNA fragment that starts at the beginning of the first contig, and ends at the end of the last. **CHEM 441** students know the order should follow (and can provide this information to you and to your eventual software). Each contig comes in two pieces:

1. FASTA format for the DNA string in the alphabet of nucleotides.

2. Coding region infomration in the GFF format.

The consecutive contigs have **overlaps**, which allow us to "merge" them together.

The goal of your software is to:

- use the implementation of the Boyer-Moore algorithm to find the largest overlap between each pair of consecutive contigs;

- merge all contig strings into a single string, taking care of the overlaps;

- verify all coding region data in the overlapping regions (see below);

- assemble a single GFF file containing coding region information for the new assembled contig.

Some quick notes about the tasks.

String matching. Your input strings come in the alphabet of nucleotides. There is, however, one caveat. Some contigs contain a fifth character in the DNA string description: `"N"`. For the purposes of string matching, `"N"` matches any other character in the nucleotide alphabet (as well as another `"N"` itself). For example, a string `"NAN"` can match the following strings: `"CAT", "AAA", "TAT", "TAC", "TAG", "GAT"` and more.

To correctly handle the occurrences of `"N"` I recommend that instead of using the `==` for comparison of individual characters, you create a function/method `nucleotideMatch(char c1, char c2)` which returns `true` if the characters match and `false` if they don't. You can incorporate dealing with `"N"` in this function. (Please note also, that your generic implementation of Boyer-Moore should not do this. This is a change to make specifically for the **CHEM 441** part of the assignment).

Coding region verification. The overlapping portions of the contigs may contain coding regions (the overlaps are expected to be relatively large). The goal of your program is to verify that the descriptions of the overlapping regions match, i.e., that the GFF files for both contigs contain the same description of the coding region: starting at the same position and ending at the same position as well. Your code must be able to handle the special case of when one contig ends in a coding region and the next contig starts in a coding region – the two coding regions need to be merged.

Coding region assembly. The GFF files contain coordinates relative to the character positions in the specific contigs. Your program will produce a single DNA string that merges multiple contigs together. You must also recompute the coordinates of the coding regions for all contigs that follow the first one, as the coordinates of the coding regions will shift.

Use of Boyer-Moore Algorithm to find largest overlap

The heart of the problem you need to solve to help **CHEM 441** students with their assignment is the discovery of the maximal overlap between a pair of strings $S_1 = s_1 \dots s_n$ and $S_2 = p_1 \dots p_m$, i.e., to find the largest such $k \le \min(m, n)$, that $p_1 = s_{n-k+1}, p_2 = s_{n-k+2}, \dots p_k = s_n$:

```
S1 =  s_1 s_2 ... s_n-k s_n-k+1 ... s_n

S2 =                    p_1     ... p_k p_k+1 ... p_m
```

A variation of Boyer-Moore algorithm allows you to find this overlap in linear time. Here is how to modify the algorithm:

- **Ignore the bad character rule.** You do not need the bad character rule, as it will stop being useful after just a few steps (our alphabet is too small for it to matter).
- **Start by aligning S_2 with with end of S_1.** and going right.
- After each mismatch invoke shift suggested by the good suffix rule. **Do not restart** the matching (as you are not trying to find the match for the entire S_2 anymore), rather consider the matching process from the point where you stopped.
- Stop when you reach the beginning of S_2. The position in S_1 that you align to the first character in S_2 at that moment will be the beginning of the largest overlapping region between S_1 and S_2.

Submission Instructions

These instructions are for your graded deliverables for **CSC 448**. **CHEM 441** students have their own set of deliverables: they rely on being able to run your software to produce them.

Use handin to submit all your files. Make certain you have a `README` file that accompanies your submission and explains how to set it up and to run it.

Use the following command to submit:

```
$handin dekhtyar 448-lab3-1 <files>
```