DNA Sequencing Part B

The Human Genome Project was a landmark 13 year project to map the human DNA. The project made it possible for many advances in medicine. It is also useful for other fields, such as agriculture and anthropology.

The basic process used was to extract the DNA from the cells of a person. Unfortunately, the whole DNA can’t be extracted in one piece. It is usually broken up into a very large number of small pieces. The small pieces are processed in a sequencer machine capable of detecting the strand of molecules. The result of processing one strand then is a sequence of letters describing the type and order of molecules in the strand. The strands only contain four bases: adenine (A), guanine (G), cytosine (C), and thymine (T). So, strands are represented by a text string containing the letters AGCT.

In order to recreate the order of the full DNA strand, these pieces must be stitched together in the correct order. To do this, the ends of two incomplete DNA strands are compared to see if they match when overlapped. If so, it is possible that these DNA strands are two pieces of a larger whole. The strands are matched and merged repeatedly until the full DNA string is reproduced.

Here is an example of two DNA strands that match when overlapped:

| Strand 1: ACGGACATAGTCATT | Strand 2: CATAGTCATTTCATG | Combined: ACGGACATAGTCATTTCATG |

Assignment

In this step of the assignment, you will implement the match and merge algorithm of DNA sequencing. The algorithm’s input is an ordered collection of strands. The strands will be in ascending order. The algorithm repeatedly creates another generation of strands until there is no change in the collection.

A new generation is created by the following process. Each strand in the old generation is used as the “left hand” side of overlap tests with every other strand in the old generation. The strand that has the longest overlap will then be used to merge with the current strand, and the resulting strand will be added to the new generation. After all new strands have been collected, the collection is sorted in ascending order, and duplicate strands are removed. Any strand that is completely a right-hand overlap with another strand in the generation must be removed.

In the case of two overlaps of the same size, the first overlap is used. In the case of no overlaps, the current strand moves to the next generation as is.

Addendum to the Algorithm

In order to prevent infinite merging with some cases of inputs, we adopt this modification to the algorithm. For an overlap to be considered for a merge, the left hand strand and the right hand strand must not contain any of the same original (generation 0) strands. For example, if the left hand strand was made by a merge between the original strands #1 and #5, and the right hand strand was made by a merge between the original strands #3 and #5, then their overlap should not be considered.

It is suggested that this be implemented by adding to the `Strand` class the ability to record the unique identifiers of all original strands that were used to create it. For example, you could add a `setUniqueId()` method to set the id of each of the original strands. Then, in the `merge` method, assign the resulting strand to have the union of ids from the merged strands. Finally, in the `overlap` method, do not allow the overlap if the two strands have any unique identifiers in common.

Requirements

- Your algorithm must be declared in `sequence.h` and implemented in `sequence.cpp`.
- The signature for the function is `std::vector<Strand> sequence(std::vector<Strand> strands);`
- Note that this algorithm can, and must, use the `std::vector` class from the STL.
- Do not `using namespace std;` in your header file.
- A unit test program must be created. This program must exercise the sequence algorithm with various input cases to assure that the output is correct.
A **Makefile** must be provided for this assignment. At minimum, it must have a target **unittest** that will be sure your unit test program is built correctly, and then run **valgrind** on your unit test program.

**Show Off Your Work**

To receive credit for this assignment, you must upload the source code (**sequence.h**, **sequence.cpp**, **Strand.h**, **Strand.cpp** and your unit test file(s)) and the **Makefile** to the CIT submission system linked at the top of the page.

Additionally, the programs must build and run. In addition to your unit tests, your **sequence** function will be tested by a grading unit test. Any incorrect performance or memory errors will be counted against the assignment score.