Homework #4

Due November 19th, 11:59pm

Each homework submission must include:

- An archive (.zip or .gz) file of the source code containing:
  - The makefile used to compile the code on Monsoon **(5pts)**
  - All .cpp and .h files **(5pts)**
- A full write-up (.pdf of .doc) file containing answers to homework's questions **(5pts)**, including the exact command line needed to execute every subproblem of the homework

The source code must follow the following guidelines:

- No external libraries that implement data structures discussed in class are allowed, unless specifically stated as part of the problem definition. Standard input/output and utilities libraries (e.g. math.h) are ok.
- All external data sources (e.g. input data) must be passed in as a command line argument (no hardcoded paths within the source code **(5pts)**.
- Solutions to sub-problems must be executable separately from each other. For example, via a special flag passed as command line argument **(5pts)**

For this homework, you will use the *query dataset* located on Monsoon: **/common/contrib/classroom/inf503/human_reads_2_trimmed.fa**.

For this homework, you will also need to use the *subject dataset* (human genome assembly that you used in HW#1). Recall that it is located at: **/common/contrib/classroom/inf503/genomes/human.txt**

- This file contains multiple scaffolds that comprise the human genome
- The genome is in FASTA format (see insert)
  - The headers are unique and always begin with the ">" character. These can be discarded for this homework. Each line of genome file is exactly 80 characters long (plus carriage return character)
  - The genomic sequences consist of the following alphabet {A, C, G, T, N}

```
>568815281-9606
CTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCT
AACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAA
CCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACC
CTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCT
AACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTTAACCCTAACCCTAACCCTAACCCTA
ACCCTAACCCTAACCCTAACCCTCACCCTTCTAACTGGACTCTGACCCTGATTGTTGAGGGCTGCAAAGAGGAA
GAATTTTATTTACCGTCGCTGTGGCCCCGAGTTGTCCCAAAGCGAGGTAATGCCCGCAAGGTCTGTGCTGATCAGGACGC
AGCTCTGCCTTCGGGGTGCCCCTGGACTGCCCGCCCGCCCGGGTCTGTGCTGAGGAGAACGCTGCTCCGCCTCCGCGGTA
CTCCGGACATATGTGCAGAGAAGAACGCAGCTGCGCCCTCGCCATGCTCTGCGAGTCTCTGCTGATGAGAACACAGCTTC
ACTTTCGCAAAGGCGCAGCGCCGGCGCAGGCGCGGAGGGGCGCGCAGCGCCGGCGCAGGCGCGGAGGGGCGCGCCCGAAC
CCGAACCCTAATGCCGTCATAAGAGCCCTAGGGAGACCTTAGGGAACAAGCATTAAACTGACACTCGATTCTGTAGCCGG
CTCTGCCAAGAGACATGGCGTTGCGGTGATATGAGGGCAGGGGTCATGGAAGAAAGCCTTCTGGTTTTAGACCCACAGGA
AGATCTGTGACGCGCTCTTGGGTAGAGCACACGTTGCTGGGCGTGCGCTTGAAAAGAGCCTAAGAAGAGGGGGCGTCTGG
AAGGAACCGCAACGCCAAGGGAGGGTGTCCAGCCTTCCCGCTTCAACACCTGGACACATTCTGGAAAGTTTCCTAAGAAA
```

Problem #1 (of 2): Needleman Wunch – aka doing it the hard way

Create a class called ***Queries_NW*** The purpose of the class will be to contain a dataset of genomic sequences (queries) and all of the functions needed to operate on this set. Use the **2D array** data-structure to store the genomic fragments of a given size. The class will be using the Needleman Wunsch algorithm to conduct fuzzy searches within its fragments. Assume that the NW algorithm will use a genomic similarity scoring matrix (+2 for match, -1 for mismatch) and a gap penalty (-1).

At minimum, the class must contain (**5pts**):

- A constructor
- A destructor
- A Needleman Wunsch function to compare two n-mer sequences, returning the similarity score of the best alignment
- A function to search for a given n-mer within the Queries_NW class, returning the best possible match (based on similarity score.

A. **(15 pts) Searching speed.** Store all fragments of the *query dataset* in your Queries_NW class. **Randomly** pick 10K, 100K, 1M n-mers of the *subject dataset* to conduct fuzzy searching within the query dataset using NW algorithm.

   - For each of your searches (10K, 100K, and 1M), how many 'hits' with up to 2 mismatches did you find?
   - For each of your searches (10K, 100K, and 1M), how long did the search take?
   - How long would the search take for the entire subject dataset?

Special note: Depending on the speed of your alignment implementation, this homework may take hours or days to complete. The goal is to get a sense for how slow these 'optimal' alignment algorithms are… for the explicit purpose of establishing a baseline to be able to compare the improved algorithms and data structures we will be discussing later in the course. Please be aware of this substitute smaller benchmarks (e.g. 1K, 10K, 100K) as appropriate.

B. **(15 pts) Searching speed**: Store all fragments of the *query dataset* in your Queries_NW class. Generate 10K, 100K, and 1M **completely random (NOT FROM ANY DATASET)** n-mers to conduct fuzzy searching within the query dataset using NW algorithm.

   - For each of your searches (10K, 100K, and 1M), how many 'hits' with up to 2 mismatches did you find?
   - For each of your searches (10K, 100K, and 1M), how long did the search take?

Problem #2 (of 2): Having a BLAST

Create a class called ***Queries_BL*** The purpose of the class will be to contain a dataset of genomic sequences (queries) and all of the functions needed to operate on this set. Use the **2D array** data-structure to store the genomic fragments of a given size. The class will be using the BLAST algorithm to conduct fuzzy searches within its fragments. Use word size of 11 for initial seed matching within the Blast algorithm and the Needleman Wunch to perform seed extension.

At minimum, the class must contain (**5 pts**):

- A constructor
- A destructor
- A Needleman Wunsch function to compare two n-mer sequences, returning the similarity score of the best alignment
- A function to search for a given n-mer within the Queries_BL class, returning the best possible match (based on similarity score.

A. **(20 pts) Searching speed.** Store all fragments of the *query dataset* in your Queries_BL class. **Randomly** pick 10K, 100K, 1M **character long segment** of the *subject dataset* to conduct fuzzy searching within the query dataset using BLAST algorithm. **Iterate through all possible n-mers of your character segment when you are conducting your search.**

   - For each of your searches (10K, 100K, and 1M), how many 'hits' with up to 2 mismatches did you find?
   - For each of your searches (10K, 100K, and 1M), how long did the search take?
   - How long would the search take for the entire subject dataset?

B. **(15 pts) Searching speed**: Store all fragments of the *query dataset* in your Queries_BL class. Generate **completely random** 10K, 100K, 1M **character long segment** to conduct fuzzy searching within the query dataset using BLAST algorithm. **Iterate through all possible n-mers of your character segment when you are conducting your search.**

   - For each of your searches (10K, 100K, and 1M), how many 'hits' with up to 2 mismatches did you find?
   - For each of your searches (10K, 100K, and 1M), how long did the search take?
   - How does that compare with the benchmarks from problem 1, part B.