Homework #3

Due March 17th, 11:59pm

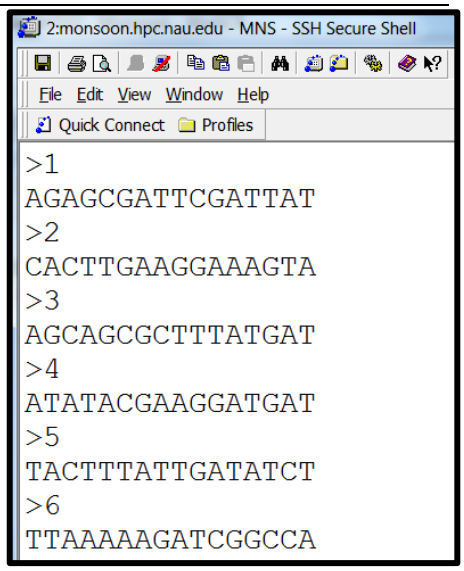
Each homework submission must include:

- 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 readme.txt file outlining all modules (if any) needed for the execution of the code and the exact command lines needed to answer homework's questions **(5pts)**
- A full write-up (.pdf of .doc) file containing answers to homework's questions **(5pts)** – screenshots of code output are ok.

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 and time.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.
- As appropriate, 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 High Throughput Sequence reads dataset located on Monsoon: **/common/contrib/classroom/inf503/hw3_dataset.fa** (see insert). Note that the header (line with ">" at the beginning) is in a format that is different from HW#1 and HW#2. For this assignment it will be safe to discard all headers.
- You will also need to use the genome sequence for Bacillus anthracis bacterium (same as HW#2) located at: **/common/contrib/classroom/inf503/test_genome.fasta**
  - This genome file contains a header (denoted by '>') followed by ~5.2 million characters of its genomic code (alphabet A, C, G, T)
  - Please be aware that the genome is spread across multiple lines of the file



Safe assumptions:

- All sequence fragments in the read set are exactly 16 characters long and that the alphabet is strictly {A, C, G, T} (no N's). This has impact on your radix calculations
- The genome sequence has no N's. This simplifies your search function.

Problem #1 (of 2): Fun with direct access arrays

Create a class called ***FASTAreadset_DA***. The purpose of the class will be to contain a FASTA read set (similar to homeworks #1 and #2) and all of the functions needed to operate on this set. Use the **direct access hash table** data-structure to store the genomic sequences of the given read dataset (hint: use an array of Boolean values – bool[] for your hash table). You will need to read in the genomic sequence fragments (feel free to ignore / discard all headers), covert them to a radix notation number (hint: try using an **unsigned int** to store the radix value), and flip the proper Boolean in the hash array to TRUE. If the Boolean is already "ON" (i.e. you are seeing a duplicate fragment), you'll need to record this 'collision'.

At minimum, the class must contain:

- A constructor
- A destructor
- A function to **search** the hash table for a given 16-mer sequence
- A function to **insert** a given 16-mer sequence into the hash table
- Private variables to store the total # of collisions and # of elements stored in the array

A. **Getting started**: read in the read data set into your data structure

- What is the size of your hash table?
- How many collisions did you observe?
- How many unique sequences did you observe (number of "ON" Boolean values)?
- What is the load (α_T) in your hash table?

B. **Search time in direct access arrays**: read in the genome sequence provided above, iterate through all 16-mers found in the genome, and use them to query the read set (similar to what you did in HW#2, problem 2B).

- How many genome 16-mer fragments were found in your read set?
- How long did it take to complete the entire search process (all 16-mers)?

Problem #2 (of 2): The hash table with chaining

Create a class called ***FASTAreadset_Chain***. Use the **hash table** data-structure to store the genomic sequences of the given read dataset (hint: you will need to provide the size of the hash table). If you have a duplicate sequence fragment or a duplicate hash value, use chaining method to resolve collisions. Resizing is optional - you can hard-code the proper hash table size through the constructor. Use Radix / division scheme for hash function implementation.

At minimum, the class must contain:

- A constructor
- A destructor
- A function to search the hash table for a given 16-mer sequence
- A function to insert a given 16-mer sequence into the hash table
- A private variable to set the hash table size
- A private variable to count the number of collisions during hash table creation

A. **Assessing the impact of the hash table size.** For this you will need to set the hash table to a fixed value (**m**, see below) and read in the read set to populate the hash table. Set the size of your hash table (***m***) to 10 thousand, 100 thousand, 1 million, and 10 million elements.

- For each of your 4 hash table sizes, how many collisions did you observe while populating the hash?
- For each of your 4 hash table sizes, how long did it take you to read the sequence fragment file?
- Do the results make sense? Explain.

B. **Searching in the chain-linked hash table.** Set the hash size to 10,000,000 and populate it using the read set. Read in the genome, iterate through all 16-mers found in the genome, and use them to query the read set (similar to what you did in HW#2, problem 2B).

- How many genome 16-mer fragments were found in your read set?
- How long did it take to complete the entire search process (all 16-mers)?
- How does that compare to the direct access array search times you've implemented as part of problem 1B?