Homework #3

Due Nov 3rd, 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
AACCCTAACCCTAACCCTAACCCTAACCCTAACCCTAACCCTTAACCCTAACCCTAACCCTAACCCTA
ACCCTAACCCTAACCCTAACCCTCACCCTTCTAACTGGACTCTGACCCTGATTGTTGAGGGCTGCAAAGAGGAA
GAATTTTATTTACCGTCGCTGTGGCCCCGAGTTGTCCCAAAGCGAGGTAATGCCCGCAAGGTCTGTGCTGATCAGGACGC
AGCTCTGCCTTCGGGGTGCCCCTGGACTGCCCGCCCGCCCGGGTCTGTGCTGAGGAGAACGCTGCTCCGCCTCCGCGGTA
CTCCGGACATATGTGCAGAGAAGAACGCAGCTGCGCCCTCGCCATGCTCTGCGAGTCTCTGCTGATGAGAACACAGCTTC
ACTTTCGCAAAGGCGCAGCGCCGGCGCAGGCGCGGAGGGGCGCGCAGCGCCGGCGCAGGCGCGGAGGGGCGCGCCCGAAC
CCGAACCCTAATGCCGTCATAAGAGCCCTAGGGAGACCTTAGGGAACAAGCATTAAACTGACACTCGATTCTGTAGCCGG
CTCTGCCAAGAGACATGGCGTTGCGGTGATATGAGGGCAGGGGTCATGGAAGAAAGCCTTCTGGTTTTAGACCCACAGGA
AGATCTGTGACGCGCTCTTGGGTAGAGCACACGTTGCTGGGCGTGCGCTTGAAAAGAGCCTAAGAAGAGGGGGCGTCTGG
AAGGAACCGCAACGCCAAGGGAGGGTGTCCAGCCTTCCCGCTTCAACACCTGGACACATTCTGGAAAGTTTCCTAAGAAA
```

Problem #1 (of 1)

Create a class called ***Queries_HT*** 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 **hash table** data-structure to store the genomic fragments of a given size. The class must include the size of the hash table (m) as one of its configurable parameters. If you have a duplicate sequence fragment or a duplicate hash value, use chaining method to resolve collisions. Use Radix / division scheme for hash function implementation.

At minimum, the class must contain (**15pts**):

- A constructor
- A destructor
- A function to search the hash table for a given n-mer sequence (returning a presence / absence Boolean value should be sufficient)
- A function to insert a given n-mer sequence into the hash table
- A function to convert a given sequence to a Radix notation (use *double* or *unsigned int* data type to store the radix value)

A. **(30 pts) Assess the impact of the hash table size.** You will be making 4 hash tables with fixed sizes (***m***). Set the size of your hash table (***m***) to 1 million, 10 million, 30 million, and 60 million elements. Populate the hash table with the sequence fragments from the *query dataset*.

   - 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 populate the hash table? Do the timing results make sense? Explain.

B. **(30 pts) Searching speed**: Set the hash table size to 60 million. Populate the hash table with the sequence fragments from the *query dataset*. Read in the entire *subject dataset* into a single, concatenated character array (same way you did it in HW#1). Implement a search function which would search for 16-character fragments of the subject sequence within the Queries_HT object. Iterate through all 16-character long fragments of the *subject dataset*, searching for each one in the *query dataset*.

   - How long did it take to search for every possible 16-character long fragment of the *subject dataset* within the *query dataset*?
   - How many such fragments did you find?
   - Print the first 15 fragments of the *subject dataset* that you found within the Query_HT.