CS341 Project 3: Let's BLAST Off!

Objectives
To implement a variation of hash table and use it for BLAST
To relate data structures and algorithms to real-world problem-solving

Description
Basic Local Alignment Search Tool (BLAST) is a bioinformatics tool that finds regions of local similarity between the query sequence and the database sequences. A sequence is either a DNA sequence or a protein sequence. This project is to implement an important part of BLAST and, to simplify the implementation, to work on the DNA sequences only. (More about BLAST can be found at the National Center for Biotechnology Information BLAST website: http://www.ncbi.nlm.nih.gov/BLAST/.)

The alphabet of the DNA sequence consists of only 4 letters, A, C, G, and T. Given a query sequence *q*, a database sequence *d*, and the word size *w*, the following steps are needed:

1. Find words of length *w* (they are called *w*-mers) from *q* and create a table to store them and record the position of their occurrences.
2. Scan *d* to find if a *w*-mer in *d* is also in *q*, i.e., in the table.

In particular, *d* and *q* are character strings, *w* is an integer of course, and a word of length *w* (*w*-mer) is a substring of length *w*. The table is to be implemented as a hash table, and each item in the table has a key value and a list of positions. Horner's rule is used to convert a word into an integer key value for hashing with A=1, C=2, G=3, and T=4.

Example
Suppose *q* = "ACACAT", *d* = "GTACACGTT", and *w* = 3.

Build the table
The first 3-mer in *q* is "ACA", and its key value = $1\times4^2+2\times4+1 = ((1\times4+2)\times4)+1 = 25$.
Hashing is performed using 25 as the input to calculate the table index.
"ACA" and the position 1 (the position of a sequence starts at 1) are stored in that location.

The second 3-mer is "CAC", and its key value = $2\times4^2+1\times4+2 = ((2\times4)+1)\times4)+2 = 38$.
Hashing is performed using 38 as the input to calculate the table index (collision resolution might be needed).
"CAC" and the position 2 are stored in that location.

The third 3-mer is "ACA", and its key value = 25.
Hashing is performed using 25 as the input to calculate the table index.
Since "ACA" has been inserted into the table, the position 3 will be added to the position list.

The fourth 3-mer is "CAT", and its key value = $2\times4^2+1\times4+4 = ((2\times4)+1)\times4)+4 = 40$.
Hashing is performed using 40 as the input to calculate the table index (collision resolution might be needed).
"CAT" and the position 4 are stored in that location.

Match *w*-mers
Use the 3-mers in *d* one at a time to find its location(s) in *q* using the hash table.
Here is the result:
GTA: not found
TAC: not found
ACA: 1, 3
CAC: 2
ACG: not found
CGT: not found
GTT: not found

(Questions:
1. How many *w*-mers are there in *q*?
2. How many *w*-mers are there in *d*?
3. How is the size of hash table determined?)

Requirements
1. Provide a prime number for the hash table size and hash function as an input in addition to *q, d,* and *w*.
2. The hash function is *h*(*x*) = *x* mod *table_size*.
3. The table size should be greater than the number of *w*-mers in *q*. It should be used as the parameter for one of the constructor to set the array size and the hash function. The other (default) constructor uses a constant 101.
4. The hash table should contain Keyed-Items.
5. Only two operations are needed: insert() and retrieve().
6. Design the interface for the hash table by yourself and implement it.
7. Horner's rule should be used to convert a character string to an integer.
8. Incorporate modulus operations into the Horner's rule.
9. Use quadratic probing to resolve collisions. (Another option is using separate chaining.)
10. Use ADT List for the position list. Reuse the code we used to study the ADT List at the beginning of this semester instead of implementing ADT List yourself.
11. Devise an option to report either both matches and mismatches or matches only.

Test cases
1. Table size = 13, *q* = "ACACAT", *d* = "GTACACGTT", and *w* = 3.
2. (More will be provided soon.)

Programs and Report
Your programs should be written in a good style and well-documented. Your report should contain the algorithms used to implement your methods described in plain English, obstacles encountered, what you learn from this project, and any other suggestions and comments.

Submission
Email your programs, results, and report to the instructor by 11:00 PM, 5/2/2011.