Lecture #13: Thursday, 17 February 2004
Topics: Programming Project

Administrative details: The project is due by 11:59 pm on Tuesday, March 9. A milestone where you describe the details of the algorithm you plan to implement, should be handed in on Tuesday March 2, in class. We encourage you to work in groups, with a maximum of 3 students per group. Please turn in only one copy per group (code and writeup). The main communication medium for this project will be the newsgroup; this is where we'll post clarifications, explanations, and other project-related details.

Motivation & Background: Last year marks both the 50th anniversary of the discovery of the structure of Dioxy Ribonucleic acid, or DNA, and the completion of the sequencing of the human genome. To give you something both algorithmic and interesting to implement we propose a programming project that is motivated by these discoveries. Sequence alignment, that is the comparison of biological sequences, is one of the most successful applications of computer science to biology. DNA is a double stranded molecule, where each strand is a chain of nucleotides. There are four types of nucleotides (Adenine, Cytosine, Guanine, Thymine), which are commonly represented by the four letters A, C, G, and T. Because of the large size of biological data (the human genome has about 3 billion nucleotides, the mouse and rat genomes are about the same size) efficient algorithms are necessary when one is analyzing genomic sequences. In this project you will implement such an algorithm.

Needleman-Wunsch algorithm: In 1970 Needelman and Wunsch[1] published the first algorithm for sequence alignment. Their algorithm calculates the edit distance between the two strings, and is closely related to the algorithm for finding the Longest Common Subsequence described in the textbook. Given a match score c, a mismatch score d, and a gap penalty g per each gapped (skipped) letter, the edit distance between two strings is the maximum weighted sum of the number of match, mismatch, & gap scores. The Needleman-Wunsch algorithm calculates the edit distance between two substring $x_1, \ldots, x_p$ and $y_1, \ldots, y_n$ by creating an $p \times n$ matrix M, where the element $M_{i,j}$ will store the edit distance between strings $x_1, \ldots, x_i$ and $y_1, \ldots, y_j$. The matrix is updated by setting

$$M_{i,j} = \max(M_{i-1,j} - g, M_{i,j-1} - g, M_{i-1,j-1} + S(x_i, y_j))$$

Where $S(x_i, y_j)$ returns the score c if the two letters are the same, and d otherwise. Unfortunately, this approach requires $O(pn)$ time and space. There are known variants that are more memory efficient, but take $O(pn)$ time nonetheless.

[1] S.B. Needleman and C.D. Wunsch, *J. Mol. Biol.* Vol. 48 (1970), pp. 443-453.

Your task: Implement a fast global aligner, which will be called *fastmatch*, that matches a DNA sequence against a library of other sequences and finds the most closely related ones. As was described earlier, standard alignment of two genomic sequences, each of length n, takes $O(n^2)$ time. This is not sufficient when dealing with long sequences. Hence, we use "shortcuts" to create a working algorithm.

In this project, we consider the restricted problem of finding the longest common subsequence whose letters satisfy an additional condition of being part of a k-mer match (exact matching of a string of length k). Clearly, the score must be modified such that only letters that are part of a k-mer match between the two strings, can be matched to one another. For example, let $k = 2$ and consider the two sequences ACGTA and ACGA. When looking at the unrestricted version of LCS, the longest common substring is of length 4 (ACGA). The matching 2-mers are AC and CG. Therefore, the longest common substring where all letters are in a 2-mers is of length 3 (ACG). The additional A cannot be added because it is not in a 2-mer match. The above score is a loose approximation of the true distance between two biological sequences; we use it because of its simplicity.

Input Format: The input data will be in the form of FASTA files, the most common format for biological sequences. A FASTA format file contains one or more sequences, each of which starts with a header line with ">" as the first symbol. Every line between two header lines may have zero or more letters from the alphabet A,T,C,G. The number of letters per line is not limited. Your program should take as input the names of two fasta files and two command-line options: k-mer size and cutoff score, in that order. The first FASTA file is the query and will always contain one sequence. The second is the database, and can have an arbitrary number of sequences.

Output Format: A list of sequences that score above a particular threshold, identified by their header line, and their corresponding score.

An example of input and output format will be posted on the class website.

Grading criteria: Your program will be graded on correctness and speed. In particular, to get full credit, your program must be able to handle query files of length up to one hundred thousand letters long, and databases up to several million letters long, where individual sequences can be up to one hundred thousand letters in a reasonable amount of time (reasonable depends on the k-mer size, but it should be minutes rather than hours given large enough k). The k-mer size can vary from 1 (in which case you are just asked for the longest common subsequence between the two strings) to 20. You may assume that the computer on which we will be testing your submission has at least one gigabyte memory. We will grade a large portion of your project automatically, so it is important that you follow the format directions.

Items to turn in:

- The milestone write-up with the description of the algorithms you plan to use, and their time/space complexity.

- A README file describing the actual implementation (could be an extended version of the milestone write-up).
- Your code, in C or C++. You are welcome to use any standard libraries available in C or C++ (except STL). You may not download any code from the web; all code you submit besides standard libraries must be your own work.
- Makefile to create the executable fastmatch from your source code.

  Submission information for the project, sample data, and a small framework with routines to read parameters and files (in C) will be available in the near future.