605.420: Algorithms for Bioinformatics

Jack Zhan

Project 3 Analysis

05/03/2016

LCS Analysis

In the longest-common-subsequence problem, we are given two sequences X=(x1,x2 …, xm) and Y=(y1,y2 …, yn) and wish to find a maximum length common subsequence of X and Y. I approached this in 2 different way. My first way was using recursive programming. This method is shown in calLCS. The time complexity of the my approach is O(2^n) in worst case. However this method creates a recursion tree that is being solved twice. When I drew the complete recursion tree, there were many sub-problems which are solved again and again. Therefore my solution had the Overlapping Substructure property. In additional the running time of a single instance of the reclusive method was too slow. This was seen when I tried to calculate the LCS for the test case.

After my initial setback, I decided to implement the method in the textbook byy implementing a tabulation method. This is shown in calLCS2. It creates a table similar to figure 15.8 in the textbook. The time complexity of the above implementation is O(mn) which is much better than the worst case time complexity of the recursive implementation.

Below is the result of running LCS against the 4 test sequences.

LCS	ACCGGTCG ACTGCGCG GAAGCCGG CCGAA	GTCGTTCG GAATGCCG TTGCTCTGT AAA	ATTGCATT GCATGGGC GCGATGCA TTTGGTTA ATTCCTCG	CTTGCTTA AATGTGCA
ACCGGTCG ACTGCGCG GAAGCCGG CCGAA	X	GTCGTCGG AAGCCGGCC GAA	ATGCTGCG CGGAGCGG CCG	CTTGCAAG GCA
GTCGTTCG GAATGCCG TTGCTCTGT AAA	GTCGTCGG AAGCCGGCC GAA	X	TGTTGATG CCGTGCTT GTAA	CTTGAATG TGCA
ATTGCATT GCATGGGC GCGATGCA TTTGGTTA ATTCCTCG	ACCGGCGA TGCGGAAC CCG	GCTTCGGG CCGTGCTT GTAA	X	CTTGCTTA AATTG
CTTGCTTA AATGTGCA	CTGCTAAG GCA	GCTTAATG TGCA	TTGCTTAT GTGCA	X

I then tested each of these solutions and confirmed that they possible LCS solution. Below is a sample test with the LCS solution bolded.

ATT**GC**A**T**T**GC**AT**G**GG**CG**C**GA**T**GC**ATTT**GG**TTAATT**CC**TC**G**

ACCGG**T**C**G**A**CTGCGCGGA**A**GC**C**GGCCG**AA

Solution:

ATGCTGCGCGGAGCGGCCG

What I Learned

The results from above confirmed that there is always more than one way to approach a problem. I learned the hard way that my initial approach would not always work and had to spend extra time trying to correct my algorithm. My initial algorithm would error out on when trying to compute the LCS of longer sequences. Eventually I re-read the chapter and switched to a tabular method which actually ran extremely.

Efficiency:

I believe that my 2nd algorithm was extremely efficient in finding the LCS when compared to the first algorithm. For the 2nd algorithm the main routine, and at most twice every time we fill in an entry of array. There are (m+1)(n+1) entries, so the total number of calls is at most 2(m+1)(n+1)+1 and the time is O(mn). For the first algorithm, if the two strings have no matching characters, the last line always gets executed, and the time bounds are binomial coefficients, which (if m=n) are close to 2^n.

Justification for Design Decision:

It was easier to code the file reader to directly read the data of the input file into array lists. This allowed me to split reading the file and running the algorithm into separate functions. In addition I had the program directly print the result so I could check if the algorithm was running correctly. In addition, I printed the both the initial sequences and the solution to check for accuracy. Two methods to calculate LCS were created. The first was unable to run due to inefficiency.

Relevance to Bioinformatics:

This type of calculation is important in bioinformatics because LCS is the simplest form of sequence alignment which allows only insertions and deletions. This is one of the first steps in creating more advanced sequence alignment models that penalizing indels and mismatches with negative scores.