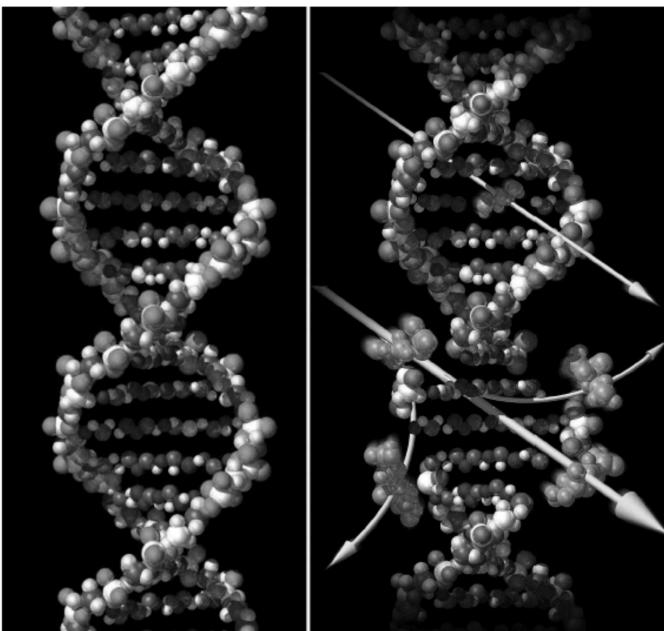


Presentation for use with the textbook, *Algorithm Design and Applications*, by M. T. Goodrich and R. Tamassia, Wiley, 2015

Dynamic Programming: Longest Common Subsequences



Effects of radiation on DNA's double helix, 2003. U.S. government image. NASA-MSFC.

Application: DNA Sequence Alignment

- ◆ DNA sequences can be viewed as strings of **A**, **C**, **G**, and **T** characters, which represent nucleotides.
- ◆ Finding the similarities between two DNA sequences is an important computation performed in bioinformatics.
 - For instance, when comparing the DNA of different organisms, such alignments can highlight the locations where those organisms have identical DNA patterns.

Application: DNA Sequence Alignment

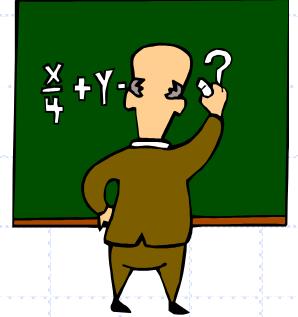
- ◆ Finding the best alignment between two DNA strings involves minimizing the number of changes to convert one string to the other.

X:	ACCGGTCGAGTGC GTC GTCGT	CGGAAGCCGGCGAA GT CG G AAGCCGGCGAA CGGAA GCCG GC C G AA GCCG GC C G AA
Y:	GT GTCGTTCGGAATGCC GTCGTTCGGAATGCC	GCTCTGTAA GCTCTGTAA

Figure 12.1: Two DNA sequences, X and Y, and their alignment in terms of a longest subsequence, GTCGTGGAAGCCGGCGAA, that is common to these two strings.

- ◆ A brute-force search would take exponential time, but we can do much better using **dynamic programming**.

The General Dynamic Programming Technique



- ◆ Applies to a problem that at first seems to require a lot of time (possibly exponential), provided we have:
 - **Simple subproblems:** the subproblems can be defined in terms of a few variables, such as j , k , l , m , and so on.
 - **Subproblem optimality:** the global optimum value can be defined in terms of optimal subproblems
 - **Subproblem overlap:** the subproblems are not independent, but instead they overlap (hence, should be constructed bottom-up).

Subsequences

- ◆ A ***subsequence*** of a character string $x_0x_1x_2\dots x_{n-1}$ is a string of the form $x_{i_1}x_{i_2}\dots x_{i_k}$, where $i_j < i_{j+1}$.
- ◆ Not the same as substring!
- ◆ Example String: ABCDEFGHIJK
 - Subsequence: ACEGJIK
 - Subsequence: DFGHK
 - Not subsequence: DAGH

The Longest Common Subsequence (LCS) Problem

- ◆ Given two strings X and Y, the longest common subsequence (LCS) problem is to find a longest subsequence common to both X and Y
- ◆ Has applications to DNA similarity testing (alphabet is {A,C,G,T})
- ◆ Example: ABCDEFG and XZACKDFWGH have ACDFG as a longest common subsequence

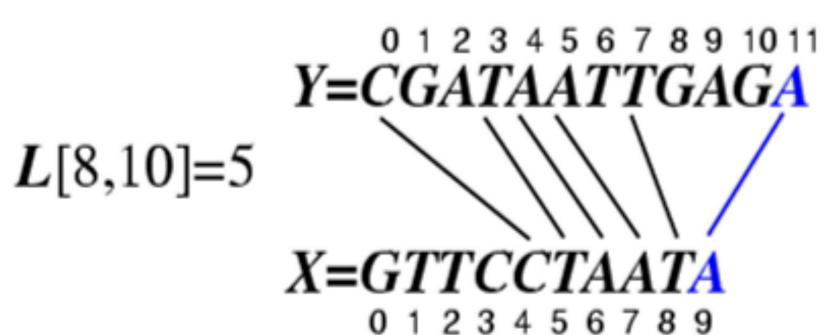
A Poor Approach to the LCS Problem

- ◆ A Brute-force solution:
 - Enumerate all subsequences of X
 - Test which ones are also subsequences of Y
 - Pick the longest one.
- ◆ Analysis:
 - If X is of length n, then it has 2^n subsequences
 - This is an exponential-time algorithm!

A Dynamic-Programming Approach to the LCS Problem

- ◆ Define $L[i,j]$ to be the length of the longest common subsequence of $X[0..i]$ and $Y[0..j]$.
- ◆ Allow for -1 as an index, so $L[-1,k] = 0$ and $L[k,-1]=0$, to indicate that the null part of X or Y has no match with the other.
- ◆ Then we can define $L[i,j]$ in the general case as follows:
 1. If $x_i=y_j$, then $L[i,j] = L[i-1,j-1] + 1$ (we can add this match)
 2. If $x_i \neq y_j$, then $L[i,j] = \max\{L[i-1,j], L[i,j-1]\}$ (we have no match here)

Case 1:



Case 2:



An LCS Algorithm

Algorithm LCS(X, Y):

Input: Strings X and Y with n and m elements, respectively

Output: For $i = 0, \dots, n-1$, $j = 0, \dots, m-1$, the length $L[i, j]$ of a longest string that is a subsequence of both the string $X[0..i] = x_0x_1x_2\dots x_i$ and the string $Y [0.. j] = y_0y_1y_2\dots y_j$

for $i = 1$ to $n-1$ **do**

$L[i, -1] = 0$

for $j = 0$ to $m-1$ **do**

$L[-1, j] = 0$

for $i = 0$ to $n-1$ **do**

for $j = 0$ to $m-1$ **do**

if $x_i = y_j$ **then**

$L[i, j] = L[i-1, j-1] + 1$

else

$L[i, j] = \max\{L[i-1, j], L[i, j-1]\}$

return array L

Visualizing the LCS Algorithm

L	-1	0	1	2	3	4	5	6	7	8	9	10	11
-1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	1	1	1	1	1	1	1	1	1	1	1
1	0	0	1	1	2	2	2	2	2	2	2	2	2
2	0	0	1	1	2	2	2	3	3	3	3	3	3
3	0	1	1	1	2	2	2	3	3	3	3	3	3
4	0	1	1	1	2	2	2	3	3	3	3	3	3
5	0	1	1	1	2	2	2	3	4	4	4	4	4
6	0	1	1	2	2	3	3	4	4	5	5	5	5
7	0	1	1	2	2	3	4	4	4	5	5	5	6
8	0	1	1	2	3	3	4	5	5	5	5	5	6
9	0	1	1	2	3	4	4	5	5	5	6	6	6

$Y = \text{CGATAATTGAGA}$
 $X = \text{GTTCTAAATA}$
0 1 2 3 4 5 6 7 8 9 10 11

Analysis of LCS Algorithm

- ◆ We have two nested loops
 - The outer one iterates n times
 - The inner one iterates m times
 - A constant amount of work is done inside each iteration of the inner loop
 - Thus, the total running time is $O(nm)$
- ◆ Answer is contained in $L[n,m]$ (and the subsequence can be recovered from the L table).