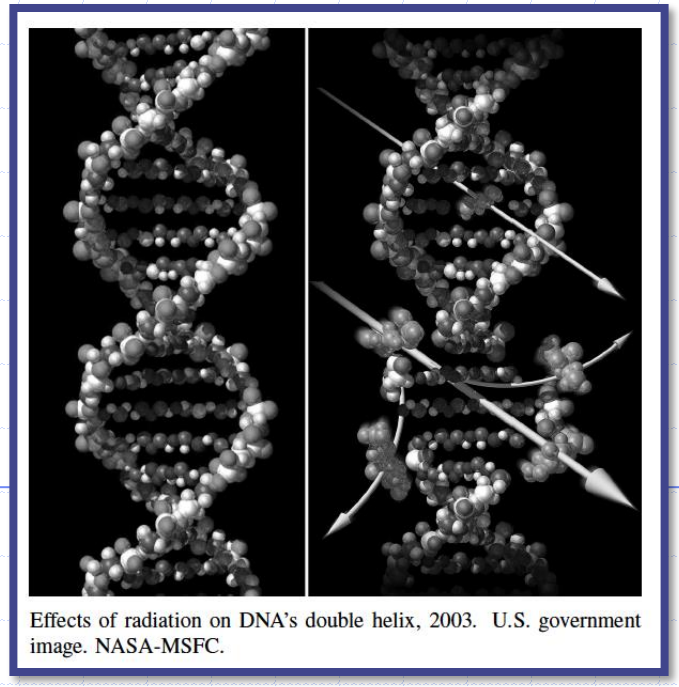
Παρουσίαση για χρήση με το σύγγραμμα, Αλγόριθμοι Σχεδίαση και Εφαρμογές, των M. T. Goodrich and R. Tamassia, Wiley, 2015 (στα ελληνικά από εκδόσεις Μ. Γκιούρδας)

Δυναμικός Προγραμματισμός: Μέγιστες Κοινές Υποακολουθίες



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

Εφαρμογή: Αντιστοίχιση ακολουθιών DNA

- Οι ακολουθίες DNA μπορούν να θεωρηθούν ως συμβολοσειρές αποτελούμενες από τους χαρακτήρες **A**, **C**, **G**, **T**, οι οποίες αναπαριστούν νουκλεοτίδια.
- Η εύρεση ομοιοτήτων ανάμεσα σε δύο ακολουθίες DNA αποτελεί μια σημαντική πράξη στη βιοπληροφορική.
  - Για παράδειγμα, όταν συγκρίνουμε το DNA διαφορετικών οργανισμών, τέτοιες αντιστοιχίσεις μπορούν να επισημάνουν τα σημεία, στα οποία αυτοί οι οργανισμοί έχουν παρόμοια μοτίβα DNA.

Εφαρμογή: Αντιστοίχιση ακολουθιών DNA

- Η εύρεση της καλύτερης αντιστοίχισης συμβολοσειρών DNA αφορά την ελαχιστοποίηση του αριθμού των αλλαγών για να μετατρέψουμε τη μία συμβολοσειρά στην άλλη.

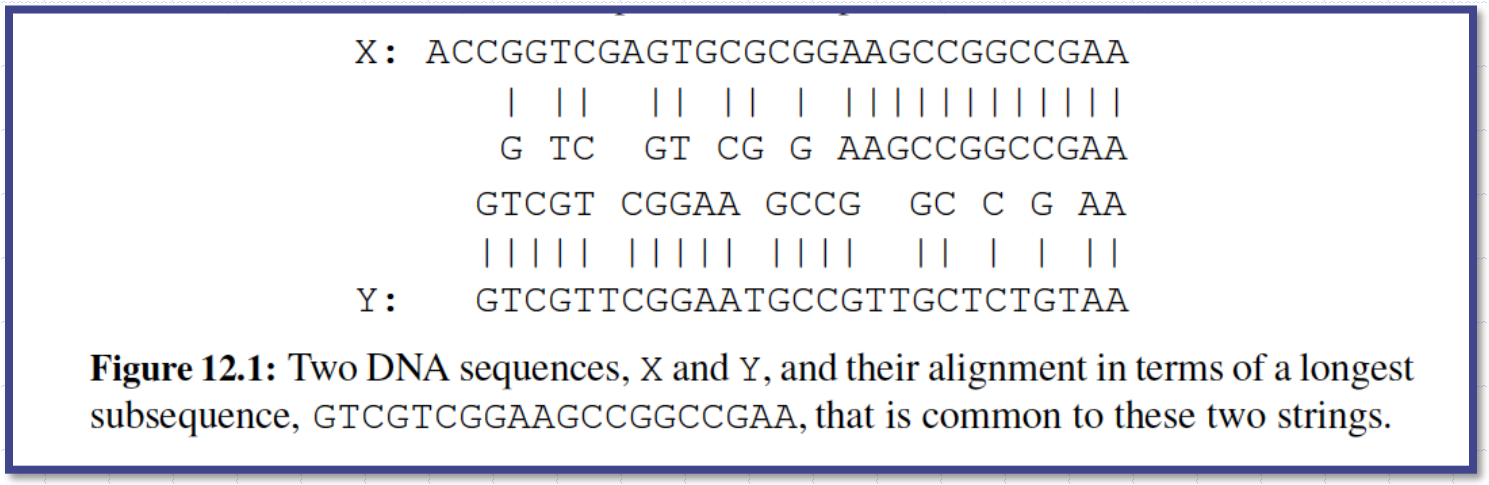
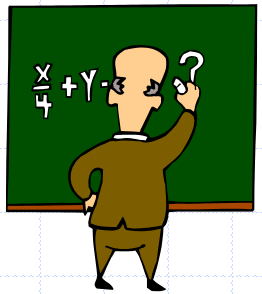


Figure 12.1: Two DNA sequences, X and Y, and their alignment in terms of a longest subsequence, GTCGTCGGAAGCCGGCCGAA, that is common to these two strings.

- Μία αναζήτηση ωμής δύναμης (brute force) θα απαιτούσε εκθετικό χρόνο αλλά μπορούμε να επιτύχουμε πολύ καλύτερα αποτελέσματα χρησιμοποιώντας **δυναμικό προγραμματισμό**.

Η γενική τεχνική δυναμικού προγραμματισμού



- Εφαρμόζεται σε προβλήματα που αρχικά φαίνεται να απαιτούν πολύ χρόνο (πιθανότατα εκθετικό), αρκεί να έχουμε:
  - **Απλά υπό-προβλήματα:** τα υπό-προβλήματα μπορούν να οριστούν χρησιμοποιώντας λίγες μόνο μεταβλητές, όπως j, k, l, m, κ.ο.κ.
  - **Βελτιστότητα υπό-προβλημάτων:** η καθολικά βέλτιστη λύση μπορεί να οριστεί με όρους βέλτιστων λύσεων σε υπό-προβλήματα.
  - **Επικάλυψη υπό-προβλημάτων:** τα υπό-προβλήματα δεν είναι ανεξάρτητα, αλλά επικαλύπτονται (οπότε πρέπει να κατασκευαστούν από κάτω προς τα πάνω).

Υποακολουθίες

- Μια **υποακολουθία** (***subsequence***) μιας συμβολοσειράς $x_0x_1x_2...x_{n-1}$ είναι μια συμβολοσειρά της μορφής $x_{i_1}x_{i_2}...x_{i_k}$, όπου $i_j < i_{j+1}$.
- Δεν είναι το ίδιο με τις υποσυμβολοσειρές!
- Παράδειγμα συμβολοσειράς: ABCDEFGHIJK
  - Υποακολουθία: ACEGIJK διότι **A**B**C**D**E**F**G**H**IJK**
  - Υποακολουθία: DFGHK διότι ABC**D**E**FGH**IJ**K**
  - Δεν είναι υποακολουθία: DAGH

Το πρόβλημα της μέγιστης κοινής υποακολουθίας

- Δεδομένων δύο συμβολοσειρών X και Y, το πρόβλημα της μέγιστης κοινής υποακολουθίας (LCS= longest common subsequence) είναι η εύρεση της μακρύτερης υποακολουθίας που είναι κοινή στη X και στη Y.
- Εφαρμόζεται στον έλεγχο ομοιότητας DNA (το αλφάβητο είναι το {A,C,G,T})
- Παράδειγμα: το ABCDEFG και το XZACKDFWGH έχουν το ACDFG ως μέγιστη κοινή υποακολουθία:
  - **A**B**CD**E**FG**
  - XZ**AC**K**DF**W**G**H

Μια «κακή» προσέγγιση στο πρόβλημα LCS

- Λύση Ωμής-Δύναμης:
  - Απαρίθμηση όλων των υποακολουθιών του X
  - Έλεγχος για το ποιες είναι επίσης υποακολουθίες του Y
  - Λήψη της μακρύτερης υποακολουθίας.
- Ανάλυση:
  - Αν το X είναι μήκους n, τότε έχει 2^n υποακολουθίες
  - Πρόκειται για αλγόριθμο εκθετικού χρόνου!

Επίλυση με Δυναμικό Προγραμματισμό του LCS

- Ορίζουμε ως L[i,j] το μήκος της μέγιστης κοινής υποακολουθίας του X[0..i] και του Y[0..j].
- Ως σύμβαση χρησιμοποιείται το -1 ως δείκτης, ώστε το L[-1,k] = 0 και το L[k,-1]=0, να υποδηλώνουν ότι το null τμήμα του X ή του Y δεν ταιριάζουν το ένα με το άλλο.
- Τότε μπορούμε να ορίσουμε το L[i,j] ως εξής:
  1. Αν $x_i=y_j$, τότε L[i,j] = L[i-1,j-1] + 1 (προσθήκη του ταιριάσματος)
  2. Αν $x_i \neq y_j$, τότε L[i,j] = max{L[i-1,j], L[i,j-1]} (δεν έχουμε ταίριασμα)

Περίπτωση 1:

Περίπτωση 2:

Ο αλγόριθμος LCS

Algorithm LCS(X,Y):

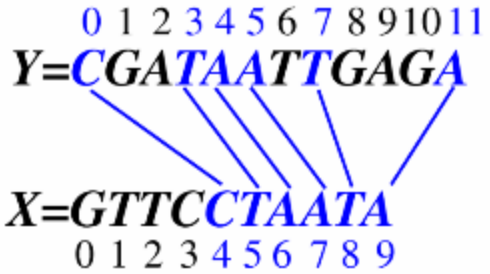
Input: Strings X and Y with n and m elements, respectively

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

```
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
```

Οπτικοποίηση του LCS

		C	G	A	T	A	A	T	T	G	A	G	A
	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
G	0	0	0	1	1	1	1	1	1	1	1	1	1	1
T	1	0	0	1	1	2	2	2	2	2	2	2	2	2
T	2	0	0	1	1	2	2	2	3	3	3	3	3	3
C	3	0	1	1	1	2	2	2	3	3	3	3	3	3
C	4	0	**1**	1	1	2	2	2	3	3	3	3	3	3
T	5	0	1	1	1	**2**	2	2	3	4	4	4	4	4
A	6	0	1	1	2	2	**3**	3	3	4	4	5	5	5
A	7	0	1	1	2	2	3	**4**	4	4	4	5	5	6
T	8	0	1	1	2	3	3	4	5	**5**	5	5	5	6
A	9	0	1	1	2	3	4	4	5	5	5	6	6	**6**



Ανάλυση του LCS

- Έχουμε δύο εμφωλευμένους βρόχους
  - Ο εξωτερικός διασχίζεται n φορές
  - Ο εσωτερικός διασχίζεται m φορές
  - Μια σταθερή ποσότητα εργασίας πραγματοποιείται σε κάθε επανάληψη του εσωτερικού βρόχου
  - Συνεπώς, ο συνολικός χρόνος εκτέλεσης είναι $O(nm)$
- Η απάντηση βρίσκεται στο L[n,m] (και η υποακολουθία μπορεί να ανακτηθεί από τον πίνακα L).