

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ & ΠΛΗΡΟΦΟΡΙΚΗΣ

|  |
| --- |
| ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΒΙΟΠΛΗΡΟΦΟΡΙΚΗ |

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2024 – 2025

ΚΟΥΡΗ ΜΑΡΙΑ

Α.Μ. 1084526

ΜΑΝΤΕΣ ΣΩΚΡΑΤΗΣ

Α.Μ. 1084526

**Περιεχόμενα**

[**Ερώτημα 1ο** 2](#_Toc196415236)

[**Υποερώτημα (i)** 2](#_Toc196415237)

[**Alpha Fold3** 2](#_Toc196415238)

[**ESMfold2** 5](#_Toc196415239)

[**Υποερώτημα (ii) (seqdump.txt)** 7](#_Toc196415240)

[**Υποερώτημα (iii)** 8](#_Toc196415241)

[**Εργαλείο MUSCLE - Multiple Sequence Comparison by Log-Expectation** 8](#_Toc196415242)

[**Εργαλείο MAFFT - Multiple Alignment using Fast Fourier Transform** 8](#_Toc196415243)

[**Εργαλείο T-COFFEE** 8](#_Toc196415244)

[**Ερώτημα 2ο** 10](#_Toc196415245)

[**Απόδειξη σχέσης** 10](#_Toc196415246)

[**Εντοπισμός μέγιστης κοινής υπακολουθίας** 11](#_Toc196415247)

[**Ερώτημα 3ο** 14](#_Toc196415248)

[**Υποερώτημα (i)** 14](#_Toc196415249)

[**Υποερώτημα (ii)** 16](#_Toc196415250)

[**Ερώτημα 4ο** 17](#_Toc196415251)

[**Ερώτημα 5ο** 17](#_Toc196415252)

[**Υποερώτημα (ii)** 17](#_Toc196415253)

[**Ερώτημα 6ο** 18](#_Toc196415254)

[**Υποερώτημα (i)** 18](#_Toc196415255)

[**Υποερώτημα (ii)** 20](#_Toc196415256)

[**BLASTP** 20](#_Toc196415257)

[**Σχολιασμός Αποτελεσμάτων** 22](#_Toc196415258)

[**Βιβλιογραφία** 22](#_Toc196415259)

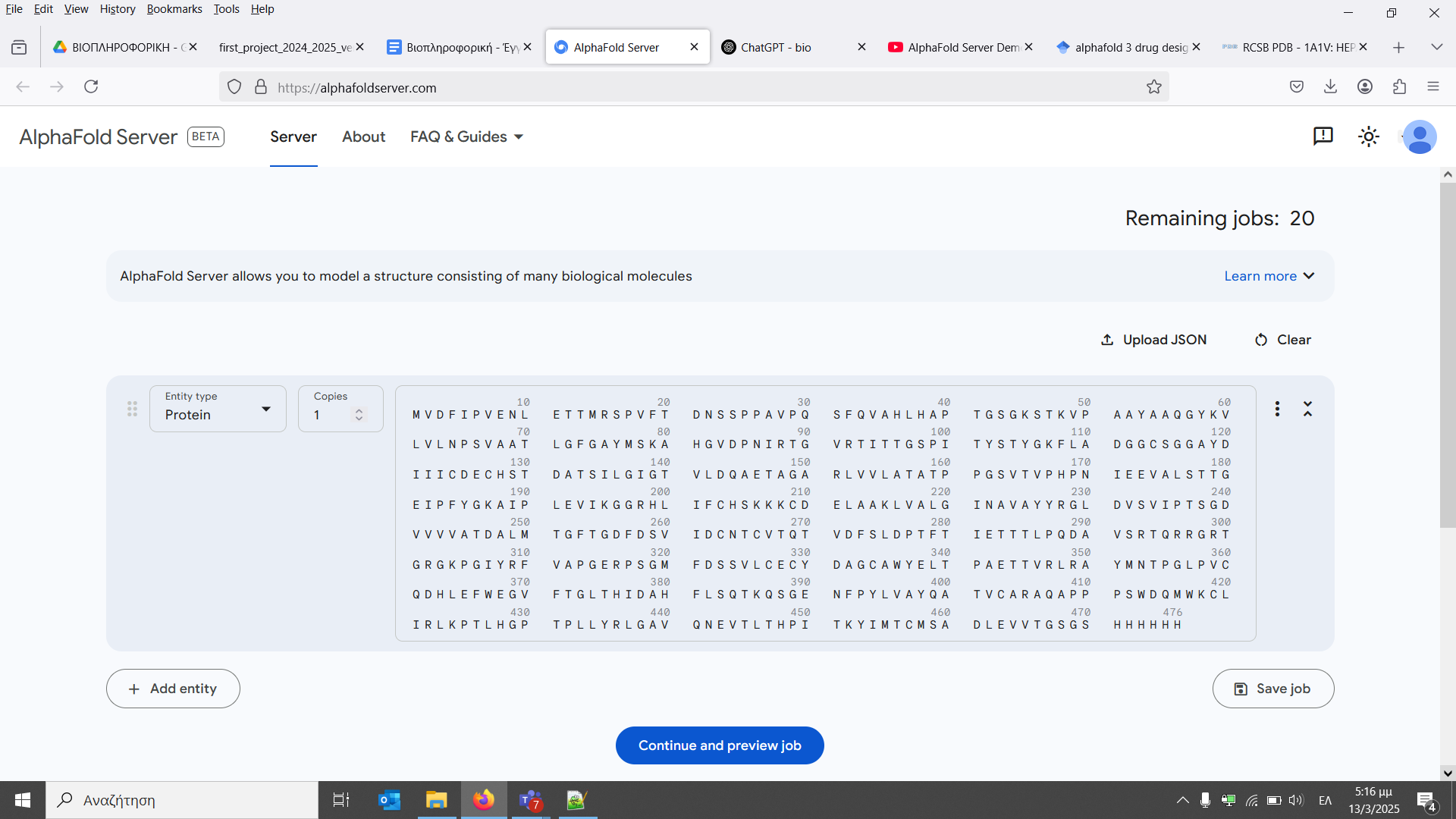
# **Ερώτημα 1ο**

## **Υποερώτημα (i)**

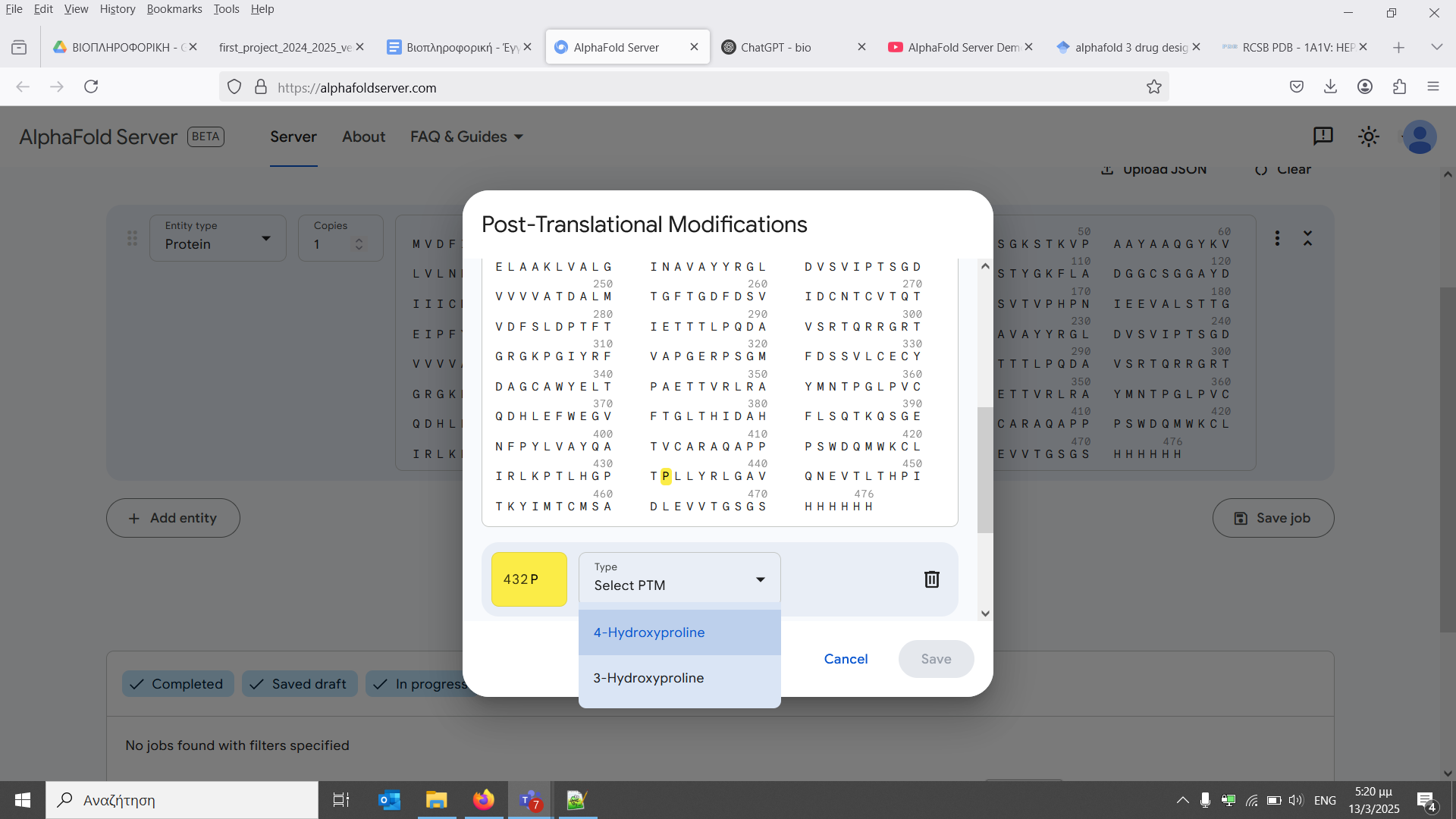
### **Alpha Fold3**

Το Alphafold είναι ένα μοντέλο τεχνητής νοημοσύνης το οποίο αναπτύχθηκε από την DeepMind της Google και την Isomorphic Labs. Το συγκεκριμένο μοντέλο επιτρέπει, με μεγάλη ακρίβεια, την πρόβλεψη της δομής μιας πρωτεΐνης, αλλά ταυτόχρονα παρέχει πληροφορίες και για τις αλληλεπιδράσεις τους με άλλα βιομόρια, όπως DNA, RNA, μικρά μόρια, ιόντα και πρωτεϊνικά κατάλοιπα. Ένα μεγάλο πλεονέκτημα το μοντέλου Alpha Fold3 έναντι των προηγούμενων είναι η ικανότητα που έχει να προβλέπει κινητικές και θερμοδυναμικές αλληλεπιδράσεις πρωτεϊνών και μορίων, γνώση η οποία μπορεί να οδηγήσει σε σχεδιασμό νέων φαρμάκων και σε καλύτερη κατανόηση της μοριακής βάσης ασθενειών [(Stampelou κ.ά., 2024)](https://www.zotero.org/google-docs/?5bKRlc), [(Desai κ.ά., 2024)](https://www.zotero.org/google-docs/?0k3wIb). Σημαντικό πλεονέκτημα του μοντέλου είναι ότι δεν απαιτεί πολλαπλές ευθυγραμμίσεις αλληλουχιών (MSA), το οποίο αυξάνει την ακρίβεια[(Manfredi κ.ά., 2025)](https://www.zotero.org/google-docs/?Sz3Wag).

Το μοντέλο Alpha Fold3 είναι διαθέσιμο προς χρήση σε online server. Για ανάλυση της λειτουργίας του χρησιμοποιείται ως παράδειγμα η πρωτεΐνη (ελικοάση) NS3 του ιού της ηπατίτιδας C (HEPATITIS C VIRUS NS3 HELICASE DOMAIN COMPLEXED WITH SINGLE STRANDED SDNA), της οποίας το αρχείο FASTA βρίσκεται στην βάση δεδομένων PDB με κωδικό καταχώρησης 1A1V. Συνεπώς δοκιμάζεται η ακολουθία της ως είσοδος:

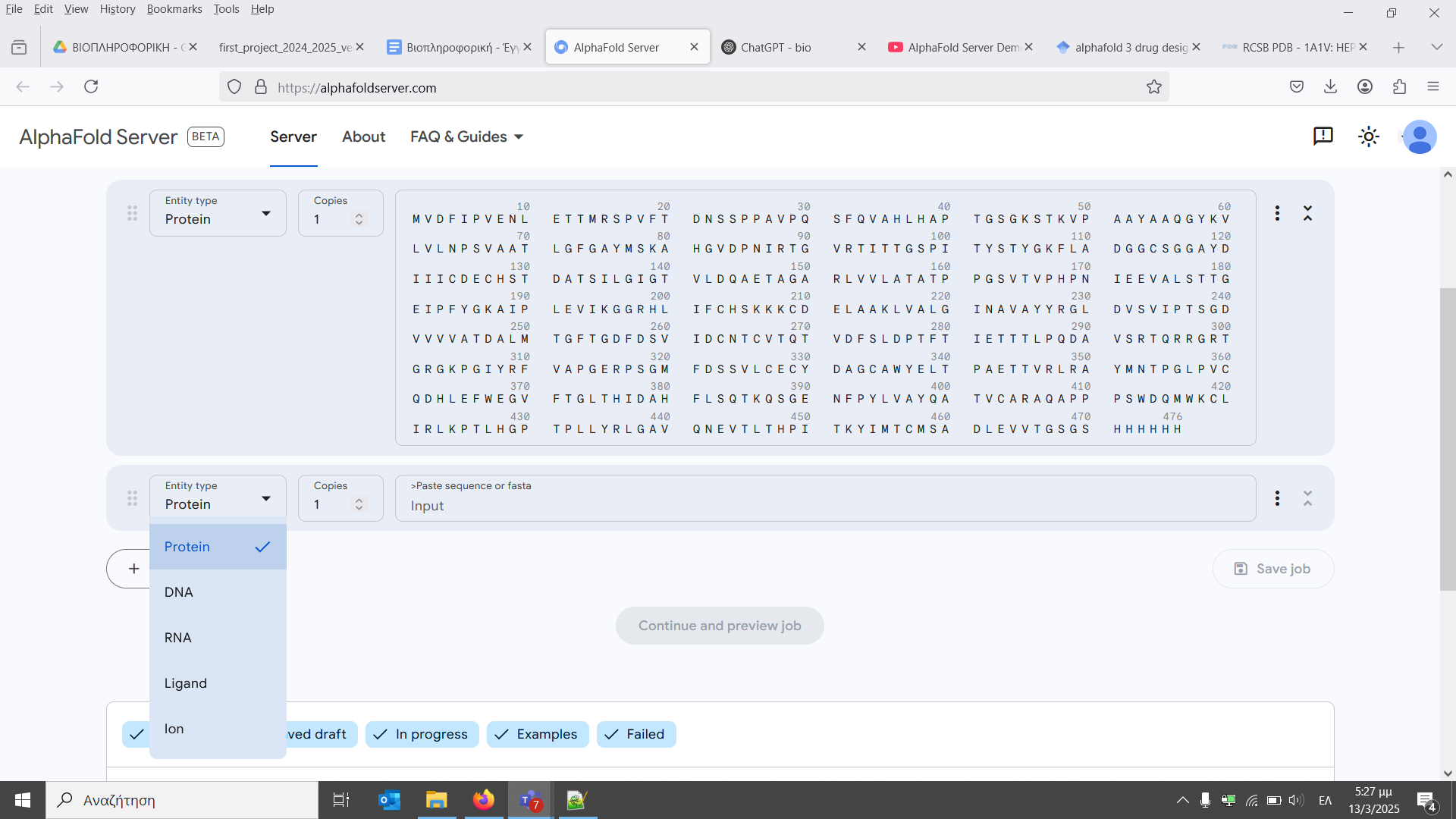


Τα νούμερα πάνω από την ακολουθία αντιπροσωπεύουν τις θέσεις των αμινοξέων στην πρωτεϊνική ακολουθία, κάτι το οποίο βοηθάει στον εύκολο εντοπισμό συγκεκριμένων στοιχείων μέσα στην ακολουθία. Με την εισαγωγή της ακολουθίας, υπάρχει η δυνατότητα να προστεθούν και άλλα χαρακτηριστικά στοιχεία, όπως για παράδειγμα οι PTMs. Οι **μετα-μεταφραστικές τροποποιήσεις (PTMs)** είναι χημικές τροποποιήσεις που υφίστανται μετά τη σύνθεσή των πρωτεϊνών (μετά τη μετάφραση του mRNA).Το συγκεκριμένο χαρακτηριστικό έχει εξέχοντα ρόλο στη ρύθμιση της λειτουργίας, της σταθερότητας, της εντοπισμού και της αλληλεπίδρασης των πρωτεϊνών (φωσφορυλίωση, ακετυλίωση, οξείδωση, κλπ.)



Εφόσον προσδιοριστούν τα χαρακτηριστικά, υπάρχει η δυνατότητα υποβολής της διεργασίας. Ως αποτέλεσμα λαμβάνεται η δομή του βιομορίου.

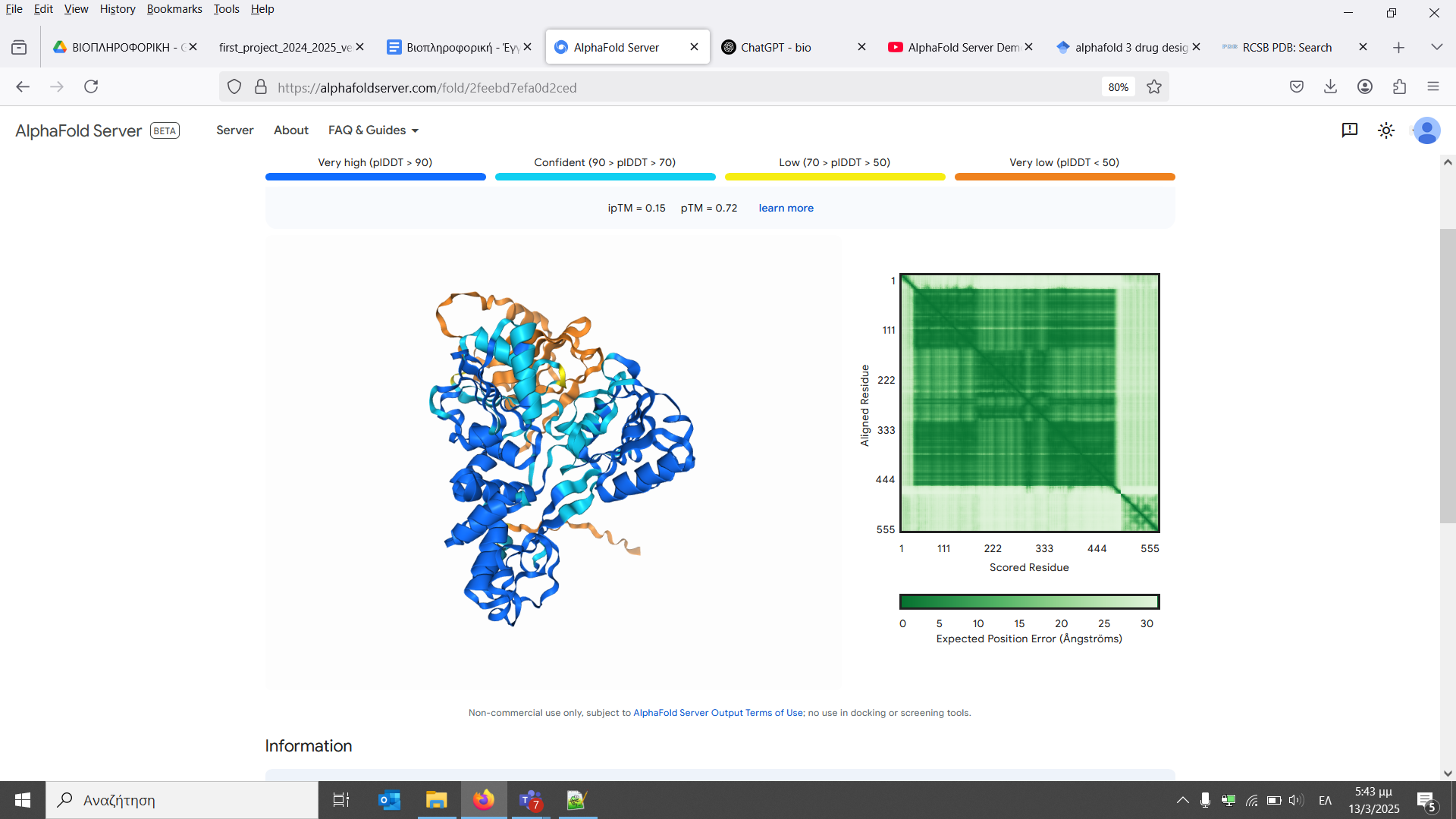
Το μοντέλο επιτρέπει στον χρήστη να προσθέσει ως είσοδο περισσότερες των μία ακολουθιών. Σε αυτήν την περίπτωση, το μοντέλο δίνει ως αποτέλεσμα τις δομές και τις αντιδράσεις - αλληλεπιδράσεις των βιομορίων μεταξύ τους.



Ως δεύτερη είσοδος τίθεται ο αναστολέας της NS4A (συμπρωτεΐνη απαραίτητη για τη λειτουργία της NS3 πρωτεάσης), της οποίας το αρχείο FASTA λαμβάνεται επίσης από την βάση δεδομένων PDB, με κωδικό καταχώρησης 1A1R (πρωτεάση NS3 + NS4A).

Μετά την συμπλήρωση της ακολουθίας της δεύτερης πρωτεΐνης και με την υποβολή της διεργασίας, το αποτέλεσμα που θα προκύψει θα είναι το σύμπλοκο NS3/NS4A, το οποίο μπορεί να διευκολύνει την μελέτη της λειτουργίας της πρωτεΐνης του HCV και στην ανακάλυψη αναστολέων για φαρμακευτικούς σκοπούς.

Από το μοντέλο προκύπτει η τρισδιάστατη δομή της πρωτεΐνης, όπου κάθε χρώμα εκφράζει την ακρίβεια της πρόβλεψης (υψηλή ακρίβεια: μπλε σκούρο, … , χαμηλή ακρίβεια: πορτοκαλί) και το γράφημα Predicted Aligned Error (PAE) αναπαριστά το **σφάλμα ευθυγράμμισης** μεταξύ των καταλοίπων της πρωτεΐνης (υψηλή ακρίβεια: σκούρο πράσινο, … , χαμηλή ακρίβεια: ανοιχτό πράσινο).



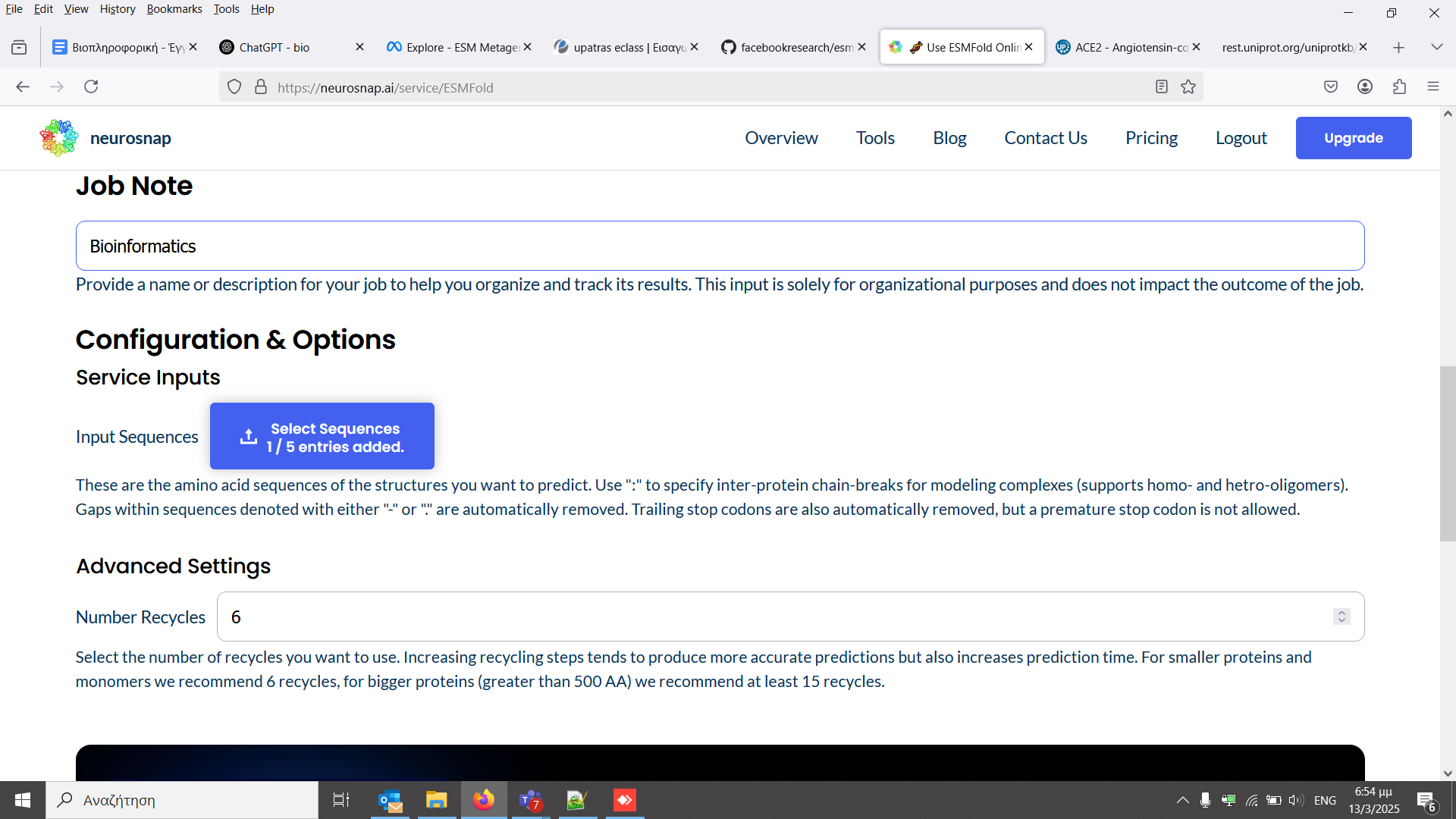
Τέλος, δίνεται η δυνατότητα λήψης των αρχείων που προέκυψαν από την εκτέλεση της διεργασίας. Ανάμεσα στα αρχεία συμπεριλαμβάνονται τα .cif αρχεία (αρχείο που χρησιμοποιείται σε βάσεις πρωτεϊνικών δεδομένων) και τα .json αρχεία (αξιοπιστία πρόβλεψης και αλληλεπιδράσεις) της πρωτεϊνικής δομής.

### **ESMfold2**

Το μοντέλο ESMfold είναι μοντέλο πρόβλεψης τρισδιάστατης δομής πρωτεϊνών που αναπτύχθηκε από την ομάδα Fundamental AI Research (FAIR) της Meta AI. Το μοντέλο Esmfold2 μπορεί να χρησιμοποιηθεί online μέσω των εργαλείων ColabFold (Google Colab), Neurosnap, UCSF ChimeraX, αλλά και τοπικά. Επιλέγεται η χρήση μέσω της πλατφόρμας Neurosnap. Η χρήση του επιτρέπει την πρόβλεψη της δομής μιας πρωτεΐνης χρησιμοποιώντας μόνο μία ακολουθία αμινοξέων, ως είσοδο. Το ESMFold2 προσφέρει γρήγορη πρόβλεψη πρωτεϊνικών δομών σε ατομικό επίπεδο χωρίς να χρειάζονται πολλαπλές ευθυγραμμίσεις αλληλουχιών (MSA), επιτρέποντας υψηλή ακρίβεια[(Manfredi κ.ά., 2025)](https://www.zotero.org/google-docs/?CPN5Ad), με χαμηλότερο υπολογιστικό κόστος από τον Alpha Fold.

Ως δείγμα χρησιμοποιείται η ακολουθία της πρωτεΐνης ACE2 (Angiotensin-Converting Enzyme 2), η οποία αποτελεί τον κύριο υποδοχέα για να εισέλθει ο ιός SARS-CoV-2 εισέρχεται στα ανθρώπινα κύτταρα. Η ακολουθία αμινοξέων υπάρχει στο UniProt σε .fasta αρχείο, με UniProt id Q9BYF1.

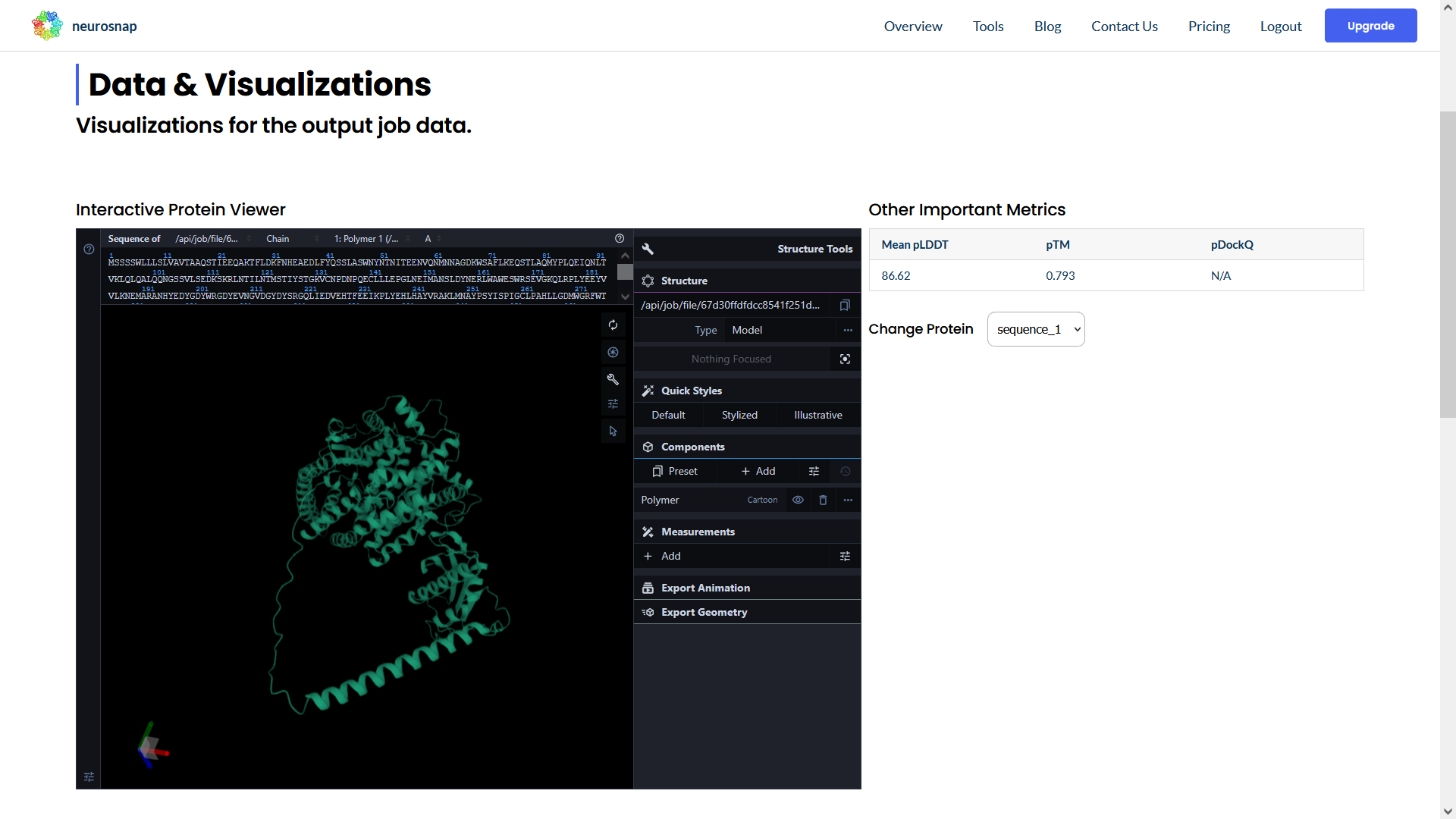
Το μοντέλο επιτρέπει την εισαγωγή περισσότερων των μία ακολουθιών, αλλά δεν θα τις επεξεργαστεί ταυτόχρονα. Το ESMFold δεν μπορεί να τις επεξεργαστεί ως αλληλεπιδρώσες πρωτεΐνες, όπως το Alphafold.



Το μοντέλο επιτρέπει στον χρήστη να επιλέξει πλήθος κύκλων που θα εκτελεστούν. Ο αριθμός των κύκλων επηρεάζει την ακρίβεια του αποτελέσματος, αλλά ταυτόχρονα και την διάρκεια της εκτέλεσης. Σε μεγάλες πρωτεΐνες ή σε πρωτεΐνες με πολύπλοκη δευτεροταγή και τριτοταγή δομή προτείνεται η χρήση περισσότερων κύκλων.

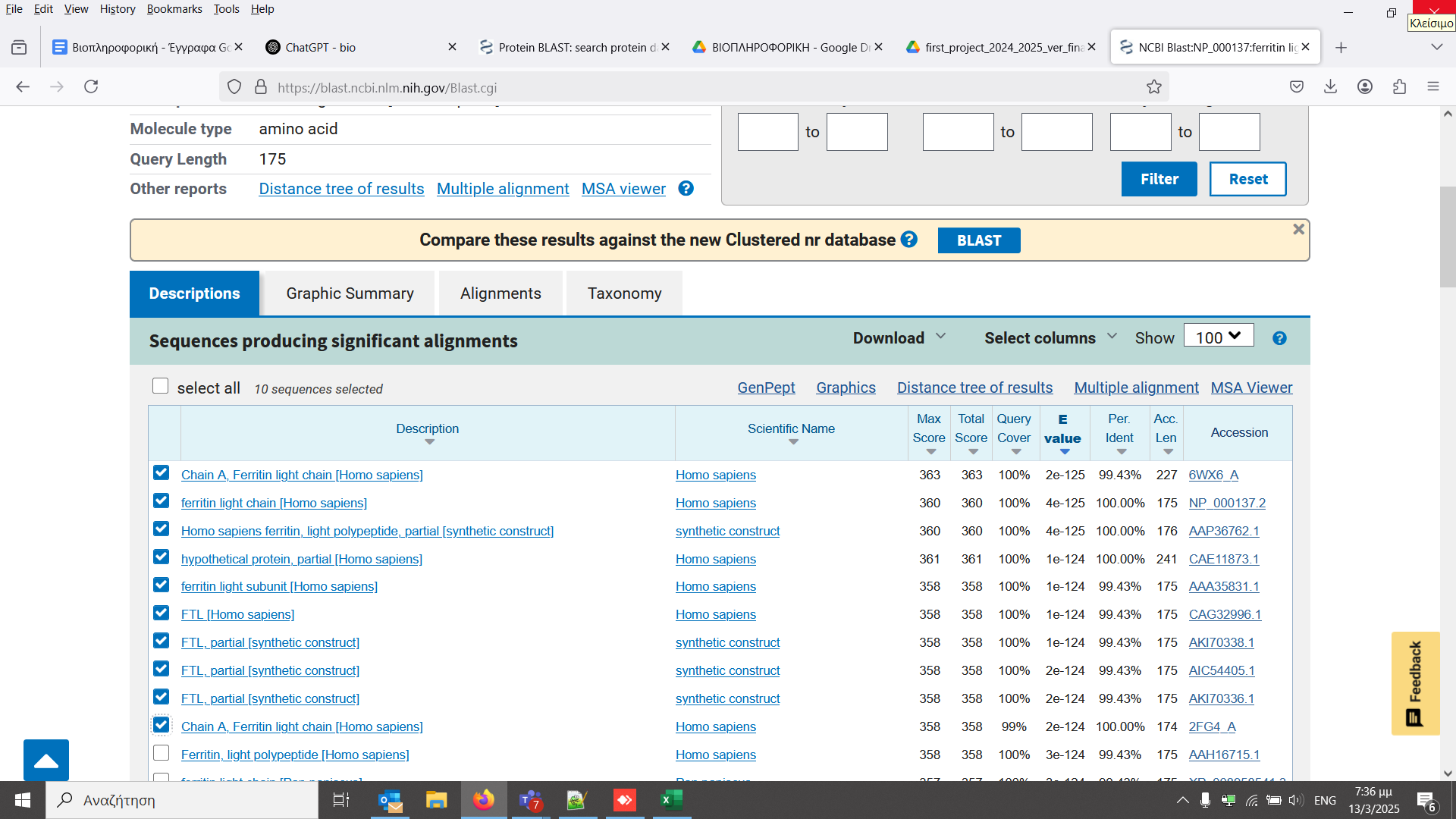
Για το συγκεκριμένο παράδειγμα επιλέγονται 8 κύκλοι. Το αποτέλεσμα που δίνεται είναι ένα .pdb αρχείο, το οποίο παρέχει πληροφορίες για την δομή και τις συντεταγμένες των ατόμων, διαγράμματα δομής (ribbon structure) που αναπαριστούν τα βασικά δευτεροταγή στοιχεία, όπως οι α-έλικες και τα β-πτυχωτά φύλλα. Τέλος, παρέχονται και οι υπολογισμοί των μετρικών:

* **pTM-score (Predicted Template Modeling Score)**: χρησιμοποιείται για σύγκριση της ομοιότητας δύο πρωτεϊνικών δομών
* **pDockQ (Predicted Docking Quality)**: χρησιμοποιείται στην πρόβλεψη πρωτεϊνικών συμπλεγμάτων για να υπολογίσει την ποιότητα του docking
* **pLDDT (Predicted Local Distance Difference Test)**: χρησιμοποιείται για να εκτιμήσει την τοπική εμπιστοσύνη της πρόβλεψης της πρωτεϊνικής δομής



## **Υποερώτημα (ii) (seqdump.txt)**





## **Υποερώτημα (iii)**

Για το συγκεκριμένο υποερώτημα χρησιμοποιούνται τα FASTA των 10 παραπάνω παραδειγμάτων, τα οποία περιέχονται στο αρχείο seqdump.txt. Σε όλες τις περιπτώσεις επιλέχθηκε η εξαγωγή των αρχείων σε μορφή ClustalW.

Όλα τα εργαλεία διαθέτουν και ειδική παραμετροποίηση η οποία επιτρέπει στον χρήστη να προσθέσει επιπλέον ελέγχους κατά την στοίχιση, αλλά να επιλέξει και το format του αρχείου που θα λάβει ως output.

### **Εργαλείο MUSCLE - Multiple Sequence Comparison by Log-Expectation**

Επιπλέον παραμετροποίηση: επιλογή format output και δέντρου εξόδου (με καμία, μία ή δύο επαναλήψεις)

*muscle-I20250320-220941-0161-16162826-p1m.aln-clustalw*

### **Εργαλείο MAFFT - Multiple Alignment using Fast Fourier Transform**

Επιπλέον παραμετροποίηση: επιλογή format output, μήτρας για πολλαπλές στοιχίσεις στην περίπτωση που η είσοδος είναι πρωτεΐνη, ποινή για άνοιγμα κενού, ποινή για επέκταση κενού, σειρά αλληλουχιών στην έξοδο, αριθμός αναδόμησης δέντρου, έξοδος του οδηγού δέντρου, μέγιστος αριθμός επαναλήψεων και εκτέλεση του FFT. Το συγκεκριμένο εργαλείο παρέχει δυνατότητες παραμετροποίησης ώστε να υπάρχει δυνατότητα αύξησης της ακρίβειας ή της ταχύτητας, αναλόγως το επιθυμητό αποτέλεσμα.

*mafft-I20250320-221641-0035-93247382-p1m.aln-clustalw*

### **Εργαλείο T-COFFEE**

Επιπλέον παραμετροποίηση: επιλογή format output, μήτρας για πολλαπλές στοιχίσεις (BLOSUM, PAM) και σειράς (αρχικής ή στοιχισμένης)

*tcoffee-I20250320-221114-0753-41116585-p1m.aln-clustalw*

Κατόπιν ελέγχου των αρχείων που προέκυψαν από τις στοιχίσεις παρατηρούνται διαφορές μεταξύ των μεθόδων. Η μέθοδος MAFFT εισάγει περισσότερα κενά για να διατηρήσει την ευθυγράμμιση των αλληλουχιών, ενώ η MUSCLE δεν προσθέτει περιττά κενά. Στις περιοχές που οι αλληλουχίες δεν ήταν ίδιες οι δύο μέθοδοι είχαν αρκετές διαφοροποιήσεις. Παρατηρείται ότι το T-COFFEE έχει μικρότερη ακρίβεια από τα άλλα δύο μοντέλα.

Για την αξιολόγηση των στοιχίσεων πέρα από οπτική αξιολόγηση, υπάρχει δυνατότητα υπολογισμού μετρικών. Οι δύο πιο διαδεδομένες μετρικές αξιολόγησης είναι το σκορ του αθροίσματος ζευγών (the sum of pairs (SP) score) και το σκορ στήλης (column score (CS)) [(Edgar & Sjölander, 2004)](https://www.zotero.org/google-docs/?FcmKkQ).

Για το SP score επιλέγεται κάθε πιθανό ζεύγος αλληλουχιών στην στοίχιση και για κάθε στήλη συγκρίνονται τα κατάλοιπα. Στην συνέχεια χρησιμοποιείται μία μήτρα αντικατάστασης για να αποθηκευτεί το σκορ κάθε ζεύγους και το συνολικό score είναι το άθροισμα των score για όλα τα δυνατά ζευγάρια και όλες τις στήλες. Μία υψηλή τιμή SP score υποδεικνύει μία καλά ευθυγραμμισμένη ακολουθία και ένα χαμηλό SP score προειδοποιεί για ασυνέπειες στην στοίχιση.

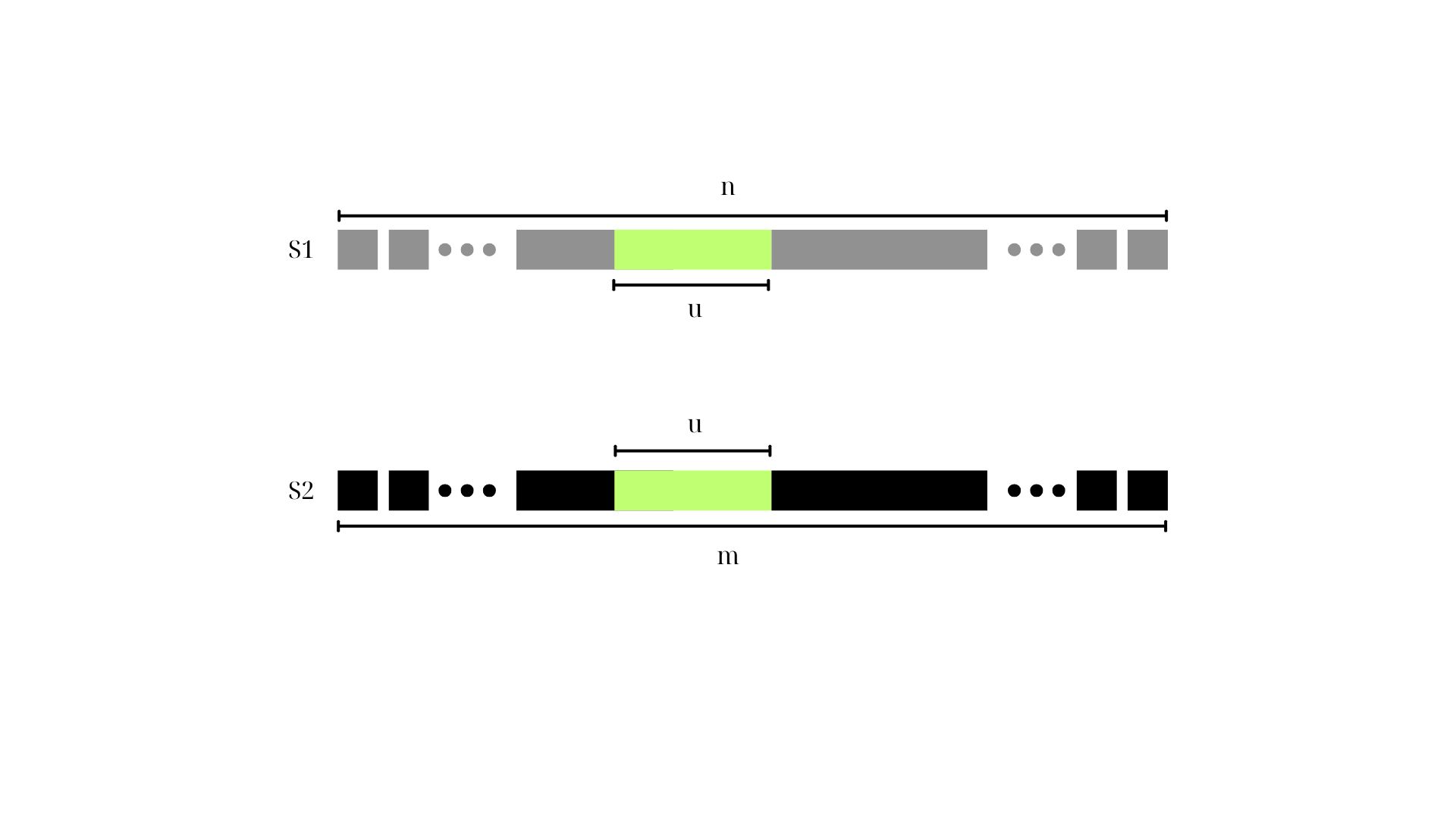
Το CS Score αξιολογεί την στοίχιση ανά στήλη και υπολογίζει πόσο καλά διατηρούνται τα κατάλοιπα σε κάθε στήλη. Για κάθε μία στήλη γίνεται έλεγχος των καταλοίπων και αν είναι ίδια η στήλη παίρνει το μεγιστο score. Το CS score προκύπτει από τον μέσο όρο των score όλων των στηλών. Ένα υψηλό CS score προϊδεάζει για ομοιόμορφες στήλες και συνεπώς καλή ποιοτικά στοίχιση, ενώ ένα χαμηλό score υποδεικνύει για χαμηλής ποιότητας στοίχιση.

Το SP Score επιβεβαιώνει την εξελικτική σχέση, ενώ το CS Score είναι εξασφαλίζει την διατήρηση συγκεκριμένων δομών και λειτουργιών. Το CS Score είναι πιο εύκολο και γρήγορο υπολογιστικά έναντι του SP Score.

# **Ερώτημα 2ο**

## **Απόδειξη σχέσης**

Έστω δύο συμβολοσειρές S1 μήκους n και S2 μήκους m, όπου u το μήκος της μεγαλύτερης κοινής υποακολουθίας των δύο συμβολοσειρών.



Επίσης ισχύει ότι:

, βάρος ένθεσης - διαγραφής

, βάρος αντικατάστασης

, βάρος ταιριάσματος

Αφού ταίριασμα (match) χαρακτήρων δεν έχει κόστος (βάρος 0), δεν αυξάνει την απόσταση. Συνεπώς δεν μπορεί να γίνει match, πρέπει είτε να γίνει διαγραφή, είτε αντικατάσταση.

Η **μέγιστη κοινή υποακολουθία** ορίζει το πλήθος των χαρακτήρων που δεν χρειάζονται ταίριασμα (match) και έτσι δεν απαιτούν επεξεργασία. Οι υπόλοιποι χαρακτήρες και από τις δύο συμβολοσειρές πρέπει να αφαιρεθούν ή να τροποποιηθούν. Έτσι η υποσυμβολοσειρά μήκους u θα παραμείνει ως έχει.

Για την ακολουθία S1:

Αν από την S1 διατηρηθεί υποσυμβολοσειρά u, έτσι κατόπιν επεξεργασίας θα πρέπει να διαγραφούν

Για την ακολουθία S2:

Αν από την S2 διατηρηθεί υποσυμβολοσειρά u, έτσι κατόπιν επεξεργασίας θα πρέπει να εισαχθουν

Έτσι επιβεβαιώνεται ότι το κόστος είναι:

Όταν το u μεγιστοποιείται το ελαχιστοποιείται.

## **Εντοπισμός μέγιστης κοινής υποακολουθίας**

Για τον εντοπισμός της μέγιστης κοινής υπακολουθίας των S1 και S2 μπορεί να εφαρμοστεί η μέθοδος του δυναμικού προγραμματισμού (Dynamic Programming). Ακολουθεί ο αλγόριθμος:

1. Ορίζεται πίνακας C διαστάσεων , όπου θα αποθηκεύει το μήκος της μέγιστης κοινής υποακολουθίας των προθεμάτων και
2. Ο πίνακας αρχικοποιείται

, για κάθε j από 0 έως m

, για κάθε i από 0 έως n

1. Ο πίνακας συμπληρώνεται με βάση την ακόλουθη σχέση:

Εάν, τότε

Διαφορετικά,

1. Το μήκος της μέγιστης κοινής υποακολουθίας είναι η τιμή
2. Για την ανακατασκευή της μέγιστη κοινή υποακολουθία, γίνεται έναρξη από το με backtracking

Εάν, τότε ανήκει στην ακολουθία και η επόμενη κίνηση είναι στην διαγώνιο

Διαφορετικά, η επόμενη κίνηση είναι στο μεγαλύτερο από και

Επαναλαμβάνεται έως ότου ή

1. Τέλος, γίνεται αντιστροφή της συμβολοσειράς που προέκυψε και το αποτέλεσμα είναι η μέγιστη κοινή υπακολουθία

Ακολουθεί κώδικας σε Python, ο οποίος βρίσκει την μέγιστη κοινή υπακολουθία και επιβεβαιώνει την παραπάνω σχέση:

Ορίζεται συνάρτηση longest\_common\_subsequence(), η οποία δέχεται ως όρισμα τις δύο συμβολοσειρές S1 και S2. Στην συνάρτηση αυτή δημιουργείται ο πίνακας που περιγράφεται παραπάνω και συμπληρώνεται με βάση τον αλγόριθμο. Αφού συμπληρωθεί ο πίνακας, ακολουθεί η εύρεση της συμβολοσειράς και η αντιστροφή της.

def longest\_common\_subsequence(s1, s2):

n = len(s1) #sequence s1

m = len(s2) #sequence s2

#Create a table to store the length

C = [[0 for \_ in range(m + 1)] for \_ in range(n + 1)]

#Fill the table

for i in range(1, n + 1):

for j in range(1, m + 1):

if s1[i - 1] == s2[j - 1]:

# If match, increment the length

C[i][j] = C[i - 1][j - 1] + 1

else:

# take the maximum from the previous subproblems

C[i][j] = max(C[i - 1][j], C[i][j - 1])

# Reconstruct the LCS

lcs = []

i, j = n, m

while i > 0 and j > 0:

if s1[i - 1] == s2[j - 1]:

# Character is part of the LCS

lcs.append(s1[i - 1])

i -= 1

j -= 1

elif C[i - 1][j] >= C[i][j - 1]:

# Move in the direction of the larger value

i -= 1

else:

j -= 1

# Reverse the sequence

lcs.reverse()

lcs\_str = ''.join(lcs)

return C[n][m], lcs\_str

Ακόμη ορίζεται συνάρτηση edit\_distance\_with\_weights(), η οποία δέχεται ως όρισμα τις συμβολοσειρές S1 και S2 και τις παραμέτρους βαρών που ορίζει η εκφώνηση. Στην συνέχεια υπολογίζει το .

def edit\_distance\_with\_weights(s1, s2, d=1, r=2, e=0): #parameters

#length of common subsequence

lcs\_length, \_ = longest\_common\_subsequence(s1, s2)

return len(s1) + len(s2) - 2 \* lcs\_length

Τέλος για λόγους ελέγχου δημιουργείται μία main(), και δύο τυχαίες συμβολοσειρές. Εκτελείται ο κώδικας.

# Example usage

def main():

s1 = "AGGCGTAGT"

s2 = "ACCTGGT"

lcs\_length, lcs = longest\_common\_subsequence(s1, s2)

distance = edit\_distance\_with\_weights(s1, s2)

print(f"String 1: {s1}")

print(f"String 2: {s2}")

print(f"Longest Common Subsequence: {lcs}")

print(f"Length of LCS: {lcs\_length}")

print(f"Edit distance D(n, m): {distance}")

print(f"Relation confirmed: {len(s1) + len(s2) - 2 \* lcs\_length == distance}")

# **Ερώτημα 3ο**

## **Υποερώτημα (i)**

s1 = "MFVFLVLLPLVSSQCVNLTTRTQLPPAYTNSFTRGVYYPDKVFRSSVLHSTQDLFLPFFSNVTWFHAIHVSGTNGTKRFDNPVLPFNDGVYFASTEKSNIIRGWIFGTTLDSKTQSLLIVNNATNVVIKVCEFQFCNDPFLGVYYHKNNKSWMESEFRVYSSANNCTFEYVSQPFLMDLEGKQGNFKNLREFVFKNIDGYFKIYSKHTPINLVRDLPQGFSALEPLVDLPIGINITRFQTLLALHRSYLTPGDSSSGWTAGAAAYYVGYLQPRTFLLKYNENGTITDAVDCALDPLSETKCTLKSFTVEKGIYQTSNFRVQPTESIVRFPNITNLCPFGEVFNATRFASVYAWNRKRISNCVADYSVLYNSASFSTFKCYGVSPTKLNDLCFTNVYADSFVIRGDEVRQIAPGQTGKIADYNYKLPDDFTGCVIAWNSNNLDSKVGGNYNYLYRLFRKSNLKPFERDISTEIYQAGSTPCNGVEGFNCYFPLQSYGFQPTNGVGYQPYRVVVLSFELLHAPATVCGPKKSTNLVKNKCVNFNFNGLTGTGVLTESNKKFLPFQQFGRDIADTTDAVRDPQTLEILDITPCSFGGVSVITPGTNTSNQVAVLYQDVNCTEVPVAIHADQLTPTWRVYSTGSNVFQTRAGCLIGAEHVNNSYECDIPIGAGICASYQTQTNSPRRARSVASQSIIAYTMSLGAENSVAYSNNSIAIPTNFTISVTTEILPVSMTKTSVDCTMYICGDSTECSNLLLQYGSFCTQLNRALTGIAVEQDKNTQEVFAQVKQIYKTPPIKDFGGFNFSQILPDPSKPSKRSFIEDLLFNKVTLADAGFIKQYGDCLGDIAARDLICAQKFNGLTVLPPLLTDEMIAQYTSALLAGTITSGWTFGAGAALQIPFAMQMAYRFNGIGVTQNVLYENQKLIANQFNSAIGKIQDSLSSTASALGKLQDVVNQNAQALNTLVKQLSSNFGAISSVLNDILSRLDKVEAEVQIDRLITGRLQSLQTYVTQQLIRAAEIRASANLAATKMSECVLGQSKRVDFCGKGYHLMSFPQSAPHGVVFLHVTYVPAQEKNFTTAPAICHDGKAHFPREGVFVSNGTHWFVTQRNFYEPQIITTDNTFVSGNCDVVIGIVNNTVYDPLQPELDSFKEELDKYFKNHTSPDVDLGDISGINASVVNIQKEIDRLNEVAKNLNESLIDLQELGKYEQYIKWPWYIWLGFIAGLIAIVMVTIMLCCMTSCCSCLKGCCSCGSCCKFDEDDSEPVLKGVKLHYT" covid

s2 = "MIHSVFLLMFLLTPTESYVDVGPDSVKSACIEVDIQQTFFDKTWPRPIDVSKADGIIYPQGRTYSNITITYQGLFPYQGDHGDMYVYSAGHATGTTPQKLFVANYSQDVKQFANGFVVRIGAAANSTGTVIISPSTSATIRKIYPAFMLGSSVGNFSDGKMGRFFNHTLVLLPDGCGTLLRAFYCILEPRSGNHCPAGNSHTSFATYHTPATDCSDGNYNRNASLNSFKEYFNLRNCTFMYTYNITEDEILEWFGITQTAQGVHLFSSRYVDLYGGNMFQFATLPVYDTIKYYSIIPHSIRSIQSDRKAWAAFYVYKLQPLTFLLDFSVDGYIRRAIDCGFNDLSQLHCSYESFDVESGVYSVSSFEAKPSGSVVEQAEGVECDFSPLLSGTPPQVYNFKRLVFTNCNYNLTKLLSLFSVNDFTCSQISPAAIASNCYSSLILDYFSYPLSMKSDLSVSSAGPISQFNYKQSFSNPTCLILATVPHNLTTITKPLKYSYINKCSRLLSDDRTEVPQLVNANQYSPCVSIVPSTVWEDGDYYRKQLSPLEGGGWLVASGSTVAMTEQLQMGFGITVQYGTDTNSVCPKLEFANDTKIASQLGNCVEYSLYGVSGRGVFQNCTAVGVRQQRFVYDAYQNLVGYYSDDGNYYCLRACVSVPVSVIYDKETKTHATLFGSVACEHISSTMSQYSRSTRSMLKRRDSTYGPLQTPVGCVLGLVNSSLFVEDCKLPLGQSLCALPDTPSTLTPRSVRSVPGEMRLASIAFNHPIQVDQLNSSYFKLSIPTNFSFGVTQEYIQTTIQKVTVDCKQYVCNGFQKCEQLLREYGQFCSKINQALHGANLRQDDSVRNLFASVKSSQSSPIIPGFGGDFNLTLLEPVSISTGSRSARSAIEDLLFDKVTIADPGYMQGYDDCMQQGPASARDLICAQYVAGYKVLPPLMDVNMEAAYTSSLLGSIAGVGWTAGLSSFAAIPFAQSIFYRLNGVGITQQVLSENQKLIANKFNQALGAMQTGFTTTNEAFRKVQDAVNNNAQALSKLASELSNTFGAISASIGDIIQRLDVLEQDAQIDRLINGRLTTLNAFVAQQLVRSESAALSAQLAKDKVNECVKAQSKRSGFCGQGTHIVSFVVNAPNGLYFMHVGYYPSNHIEVVSAYGLCDAANPTNCIAPVNGYFIKTNNTRIVDEWSYTGSSFYAPEPITSLNTKYVAPQVTYQNISTNLPPPLLGNSTGIDFQDELDEFFKNVSTSIPNFGSLTQINTTLLDLTYEMLSLQQVVKALNESYIDLKELGNYTYYNKWPWYIWLGFIAGLVALALCVFFILCCTGCGTNCMGKLKCNRCCDRYEEYDLEPHKVHVH"

mers cov

Στον κώδικα του προηγούμενου ερωτήματος προστέθηκαν οι ακολουθίες των spikes των δύο πρωτεϊνών.

Το αποτέλεσμα που προκύπτει από την εκτέλεση του κώδικα είναι το ακόλουθο:

**Longest Common Subsequence:** MVFLLLPVSSCVTTRPAGYPRSTQLFPHVSGTGTKFVNDVFANRIGTTSTSATIKPFLGVNKMFRYSANTFYSDLNFKNLRFNIDGITVLFSVDLGNFQTLYTPSSSWAAYVYLQPTFLLYIADCLLSESFVEGYSFVQEVFPTNLVFNTFSVAISNCYSLYFSYSKDLVAGIQNYKPTCIANLKLYSNKRDTEQASPCVPSTVGYYRLSELATVTLFGTGTTSKLFADTAQLELSGGVVGQVAVYDNCVPVIHALTTRSTGQTGCLGVNSECPGCASTTPRRSVASIILNSYSIPTNFSVTEITKTVDCYCGELLYGFCQALGAQDNFAVKQIPFGGFNSISSRSIEDLLFKVTADGQYDCGAARDLICAQKVLPPLDEAYTSLLAGGWTAGAAIPFAQYRNGITQVLENQKLIANFNAGQTAKQDVNNAQALKLSSNFGAISSDIRLDVEAQIDRLIGRLLVQQLRAASALAKECVQSKRFCGGHSFAPGFHVYVPANFTTIDGAPESNTVTQNITNGNGIDQELDFFKNTSPGSINSVVKALNESIDLELGYYKWPWYIWLGFIAGLAVILCCTCCLKCCCDEDEPVVH

**Length of LCS:** 592

**Edit distance D(n, m):** 1442

**Relation confirmed:** True

**Σχολιασμός αποτελεσμάτων**

Οι ακίδες των δύο πρωτεϊνών των δύο ιών έχουν **Length of LCS: 592** κοινά αμινοξέα. Αυτό υποδηλώνει πως οι δύο πρωτεΐνες έχουν μεγάλο ποσοστό κοινών δομικών περιοχών. Ακόμη η απόσταση **D(n, m): 1442** σημαίνει πώς απαιτούνται 1442 μονάδες κόστους, καθώς τα στελέχη δεν είναι κοντά. Τέλος επιβεβαιώνεται η ισχύς της παρακάτω σχέσης **Relation confirmed: True**

,

με βάση το προηγούμενο ερώτημα.

Τα βάρη διατηρούνται ίδια με το προηγούμενο ερώτημα.

, βάρος ένθεσης - διαγραφής

, βάρος αντικατάστασης

, βάρος ταιριάσματος

## **Υποερώτημα (ii)**

Το SWISS-MODEL είναι ένα εργαλείο homology modeling. Δέχεται ως είσοδο μία ακολουθία αμινοξέων και επιστρέφει models σε .pdb μορφή που προσεγγίζουν την ακολουθία. Τα μοντέλα που παρέχει ως αποτελέσματα είναι κατά ένα ποσοστό όμοια με την αρχική ακολουθία.

Κάθε μοντέλο συνοδεύεται από κάποιες αριθμητικές τιμές. Για να γίνει ορθή επιλογή του μοντέλου λαμβάνονται υπόψιν οι μετρικές.

Η μετρική **GQME** - Global Model Quality Estimate (Bonvin, 2025) αξιολογεί την ποιότητα του μοντέλου πριν αυτό κατασκευαστεί. Η τιμή κυμαίνεται από 0 έως +1 και όσο προσεγγίζει το +1 θεωρείται πιο αξιόπιστο το μοντέλο. Συνδυάζει πληροφορίες για το πόσο καλή είναι η ευθυγράμμιση (alignment) της αλληλουχίας σε σχέση με το μοντέλο.

Η μετρική **QMEANDisCo** (Studer κ.ά., 2020) Global είναι δείκτης ποιότητας του τελικού μοντέλου και μπορεί να πάρει τιμή από -4 έως +1. Όταν η τιμή προσεγγίζει το +1 το μοντέλο θεωρείται ακριβές. Για τον υπολογισμό της αξιοποιούνται στατιστικά χαρακτηριστικά από γνωστές πρωτεΐνες και αξιολογείται η «φυσιολογική» δομή.

Η τιμή **Seq Identity** (*Help | SWISS-MODEL*) υπολογίζει το ποσοστό ομοιότητα της αλληλουχίας εισόδου με την αλληλουχία του μοντέλου. Το ποσοστό κυμαίνεται από 0 έως 100% και όσο πιο κοντά είναι το 100% θεωρείται ότι το μοντέλο προσεγγίζει την πραγματική δομή της πρωτεΐνης.

Η μετρική **Coverage** (*Help | SWISS-MODEL*) αποτυπώνει το ποσοστό της ακολουθίας που εισάγεται, το οποίο μοντελοποιείται στο παρόν μοντέλο. Η τιμή της συγκεκριμένης μετρικής πρέπει να είναι υψηλή για να έχουμε μεγάλη ακρίβεια.

Η μεταβλητή **Oligo-State** δηλώνει σε ποια μορφή μοντελοποιείται η πρωτεΐνη:

* **Monomer**: μονομερές
* **Homo-trimer**: τρία ίδια μόρια ενωμένα, πολυμερές

**\*Template** είναι το PDB ID του προτύπου που χρησιμοποιείται για να απεικονιστεί το μοντέλο.**\***

### **SARS-CoV-2**

Εκτελώντας το μοντέλο για την ακολουθία του spike της πρωτεΐνης του SARS-CoV-2, προκύπτουν 6 μοντέλα. Τα μοντέλα αυτά είναι τα ακόλουθα:

|  |  |
| --- | --- |
| Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης  Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο. |  |
|  |  |
|  |  |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Oligo-State | GMQE | QMEANDisCo Global | Template | SeqIdentity (%) |
| Model 01 | Homo-trimer | 0.64 | 0.63 | 8wly.A | 77.45 |
| Model 02 | Homo-trimer | 0.67 | 0.71 | 8ajl.1.A | 80.15 |
| Model 03 | Homo-trimer | 0.63 | 0.63 | 6acd.1.A | 76.79 |
| Model 04 | Monomer | 0.71 | 0.72 | 8z6s.1.G | 94.47 |
| Model 05 | Monomer | 0.60 | 0.55 | 7fg3.1.A | 100 |
| Model 06 | Monomer | 0.41 | 0.55 | 8hxk.1.B | 91.42 |

Από τα μοντέλα επιλέγεται το **Model 04** και το .pdb αρχείο του, λόγω υψηλής ακρίβειας.

### **MERS-CoV**

Εκτελώντας το μοντέλο για την ακολουθία του spike της πρωτεΐνης του MERS-CoV, προκύπτουν τα ακόλουθα μοντέλα.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Model | Oligo-State | GMQE | QMEANDisCo Global | Template | SeqIdentity (%) |
| Model 01 | Monomer | 0.72 | 0.76 | 8ifn.1. A | 99.15 |
| Model 02 | Monomer | 0.69 | 0.72 | 7x28.1.C | 99.69 |
| Model 03 | Homo - trimer | 0.47 | 0.49 | 7sbr.1. A | 32.13 |
| Model 04 | Homo - trimer | 0.43 | 0.48 | 7ybk.1. A | 32.08 |
| Model 05 | Homo - trimer | 0.38 | 0.45 | 6acd.1. A | 31.32 |
| Model 06 | Monomer | 0.46 | 0.50 | 7wgb.1. B | 31.41 |
| Model 07 | Homo - trimer | 0.35 | 0.45 | 7ybh.1. A | 35.70 |

Από τα μοντέλα επιλέγεται το **Model 01** και το .pdb αρχείο του, λόγω υψηλής ακρίβειας.

### **Σύγκριση – DALI**

Το εργαλείο DALI αποτελεί μία βασική πλατφόρμα σύγκρισης τρισδιάστατων δομών πρωτεϊνών. Χρησιμοποιείται για την σύγκριση δύο πρωτεϊνών που έχουν παρόμοια χωρική (δομική) οργάνωση, ακόμα κι αν οι αλληλουχίες τους διαφέρουν πολύ. Αντί να συγκρίνει τις γραμμικές ακολουθίες των αμινοξέων, όπως γίνεται στην στοίχιση των ακολουθιών, μετράει τις αποστάσεις μεταξύ όλων των ατόμων του Άλφα Άνθρακα (Ca) σε κάθε πρωτεΐνη και δημιουργεί έναν πίνακα αποστάσεων. Έτσι το συγκεκριμένο εργαλείο επιτρέπει τον εντοπισμό δομικών και χωρικών συσχετίσεων, ακόμη και στην περίπτωση που οι αλληλουχίες δεν μοιάζουν.

Ως αποτέλεσμα της σύγκρισης δίνονται ένα interactive .html αρχείο που μπορεί να επεξεργαστεί ο χρήστης σε web περιβάλλον και ένα plain .txt που δίνονται τα αποτελέσματα της σύγκρισης και οι μετρικές.

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, έγγραφο

Το περιεχόμενο που δημιουργείται από τεχνολογία AI ενδέχεται να είναι εσφαλμένο.

Κατόπιν ανάλυσης των μετρικών προκύπτει ότι παρόλο οι ακίδες των δύο πρωτεϊνών έχουν χαμηλή ομοιότητα σε επίπεδο αλληλουχίας, με μόλις 28% ταύτιση. Παρόλα αυτά σε δομικό επίπεδο εμφανίζουν εξαιρετικά υψηλή ομοιότητα. Συγκεκριμένα, η σύγκριση μέσω του εργαλείου DALI ανέδειξε Z-score = 35.6, μια τιμή που υποδηλώνει ισχυρότατη δομική συσχέτιση. Η στοίχιση περιλάμβανε 932 κατάλοιπα (lali), με μέση απόκλιση (RMSD) 7.0 Å, κάτι που αποδεικνύει την διατήρηση όμοιας δομής, παρά τις τοπικές διαφοροποιήσεις που υπάρχουν.

Τα παραπάνω αποτελέσματα επαληθεύουν ότι η δομική ομοιότητα και η ομοιότητα σε επίπεδο ακολουθίας δεν είναι έννοιες αλληλένδετες, όπως θα ήταν το αναμενόμενο. Οι δύο ιοί, παρά την διαφορετική εξέλιξή τους και την διαφοροποίηση στις αλληλουχίες τους, διατηρούν παρόμοια τρισδιάστατη δομή στο spike της πρωτεΐνης τους, η οποία είναι ιδιαίτερα χρήσιμη πληροφορία. Η παρατήρηση αυτή έχει ιδιαίτερο ενδιαφέρον για τη φαρμακευτική στόχευση, καθώς κοινές δομικές περιοχές ενδέχεται να αξιοποιηθούν για ευρύτερης δράσης αναστολείς ή εμβολιαστικές πλατφόρμες.

# **Ερώτημα 4ο**

**Υποερώτημα (i)**

Στην συγκεκριμένη περίπτωση υπάρχουν δύο αλληλουχίες, οι οποίες πρέπει να στοιχίζονται ολόκληρες από την αρχή μέχρι το τέλος (καθολική στοίχιση, όχι μόνο ένα κομμάτι τους όπως στην τοπική στοίχιση) και να λαμβάνουν μπόνους στα ταιριάσματα (+1) και ποινές στα κενά ή ασυμφωνίες, αλλά με συγγενική ποινή. Δηλαδή, μεγάλη ποινή όταν ανοίγει νέο κενό (gap opening, –ρ) και μικρότερη ποινή όταν επεκτείνεται ένα ήδη υπάρχον κενό (gap extension, –σ ανά θέση). Το ρ είναι το κόστος προκειμένου να ανοίξει ένα νέο κενό (gap opening penalty), ενώ το σ είναι το κόστος προκειμένου να επεκταθεί ένα ήδη υπάρχον κενό (gap extension penalty). Το ρ θα πρέπει να είναι μεγαλύτερο από το σ, καθώς επιβαρύνεται περισσότερο αποδοτικά ο αλγόριθμος όταν χρειάζεται να ξεκινήσει ένα κενό (να διακοπεί δηλαδή η στοίχιση) σε σχέση με το να συνεχίσει ένα κενό αφού έχει ήδη ανοίξει.

Παρακάτω επεξηγείται αναλυτικά ο ψευδοκώδικας που περιγράφει τον αλγόριθμο εύρεσης της βέλτιστης καθολικής στοίχισης δύο αλληλουχιών χρησιμοποιώντας μια μέθοδο βαθμολόγησης με συγγενική ποινή ασυμφωνίας :

Η εύρεση της βέλτιστης καθολικής στοίχισης με συγγενική ποινή ασυμφωνίας στηρίζεται στη διαμόρφωση τριών διακριτών καταστάσεων, μέσω τριών πινάκων: M, X και Y. Αυτοί οι πίνακες καταγράφουν διαφορετικά είδη στοίχισης μεταξύ των δύο αλληλουχιών A και B. Ο πίνακας M[i][j] αποθηκεύει το βέλτιστο σκορ στοίχισης των πρώτων i χαρακτήρων της A με τους πρώτους j χαρακτήρες της B, όταν το τελευταίο στοιχισμένο ζεύγος είναι απλό ταίριασμα ή ασυμφωνία χωρίς κανένα κενό. Ο πίνακας X[i][j] καταγράφει την καλύτερη στοίχιση στην περίπτωση που το στοιχείο A[i] αντιστοιχίζεται με κενό (gap) στη B. Αντίστοιχα, ο πίνακας Y[i][j] αφορά την περίπτωση όπου το στοιχείο B[j] ευθυγραμμίζεται με κενό στην A.

Η διαδικασία ξεκινάει με αρχικοποίηση των πινάκων. Στο κελί M[0][0] ανατίθεται τιμή μηδέν, καθώς δεν έχει γίνει καμία στοίχιση ακόμα. Τα υπόλοιπα κελιά αρχικοποιούνται με πολύ μικρές τιμές (–∞), ώστε να μην επιλεχθούν κατά λάθος στη συνέχεια. Τα κενά στις άκρες (όπου μία αλληλουχία είναι άδεια και η άλλη έχει χαρακτήρες) έχουν ποινή που υπολογίζεται ως ρ για το άνοιγμα του πρώτου κενού, και –σ για κάθε επιπλέον επέκταση. Συνεπώς, για i κενά το κόστος γίνεται –(ρ + (i-1)σ), και το ίδιο ισχύει συμμετρικά για την άλλη αλληλουχία.

Στο βασικό στάδιο, τα κελιά συμπληρώνονται με βάση αναδρομικές σχέσεις. Όταν στοιχίζουμε χαρακτήρες (όχι κενά), το κελί M[i][j] λαμβάνει τη μέγιστη τιμή από τρεις πιθανές πορείες, είτε ερχόμενο από το M[i-1][j-1], είτε από το X[i-1][j-1], είτε από το Y[i-1][j-1], προσθέτοντας +1 αν οι χαρακτήρες A[i] και B[j] ταιριάζουν, ή –1 σε περίπτωση ασυμφωνίας. Έτσι, το M[i][j] πάντα αποτυπώνει το καλύτερο δυνατό σκορ για στοίχιση χωρίς κενό στο τέλος.

Αντίθετα, για κενό στη B (δηλαδή A[i] με κενό), το κελί X[i][j] ενημερώνεται ελέγχοντας δύο περιπτώσεις, αν ανοίγουμε νέο κενό (πληρώνουμε –ρ–σ), ή αν επεκτείνουμε ήδη υπάρχον κενό (πληρώνουμε μόνο – σ). Το ίδιο ισχύει και για το κελί Y[i][j], που αφορά κενό στην A. Αυτή η λογική εφαρμογής διαφορετικού κόστους μεταξύ ανοίγματος και επέκτασης κενού είναι χαρακτηριστικό της συγγενικής ποινής ασυμφωνίας (affine gap penalty), και διαχωρίζει τον αλγόριθμο από απλά μοντέλα με σταθερή ποινή.

Με τη σωστή συμπλήρωση όλων των κελιών, ο αλγόριθμος διασφαλίζει ότι για κάθε σημείο (i,j) έχει αποθηκευτεί το καλύτερο σκορ στοίχισης για την αντίστοιχη κατάσταση (match, gap στη B ή gap στην A). Καθώς προχωράμε προς το τέλος στο κελί (n,m), το βέλτιστο τελικό σκορ της στοίχισης υπολογίζεται επιλέγοντας το μέγιστο ανάμεσα στα M[n][m], X[n][m] και Y[n][m]. Αυτό εξασφαλίζει ότι λαμβάνουμε υπόψη όλους τους δυνατούς τρόπους τερματισμού της στοίχισης, είτε με απευθείας αντιστοίχιση είτε με κενά.

Η διαδικασία δεν σταματά στον υπολογισμό του τελικού σκορ. Για να ανακατασκευάσουμε την πραγματική στοίχιση των αλληλουχιών, εκτελείται backtracking. Ξεκινώντας από το κελί (n,m) και ακολουθούνται τα ίχνη (ποιο πίνακα και ποιο κελί οδηγήσαμε στο σκορ αυτό) και σε κάθε βήμα επιλέγεται εάν έγινε αντιστοίχιση δύο χαρακτήρων ή εισαγωγή κενού. Όταν υπάρχει μετάβαση από Μ, σημαίνει αντιστοίχιση χαρακτήρων, ενώ όταν υπάρχει μετάβαση από Χ ή Υ, σημαίνει ότι ένα κενό εισήχθη αντίστοιχα στην B ή στην A. Έτσι, δημιουργούνται οι τελικές στοιχισμένες ακολουθίες, με εισαγωγή κενών στις σωστές θέσεις ώστε να πετύχουμε το συνολικό βέλτιστο σκορ.

Δοθέντων δύο αλληλουχιών A και B μήκους n και m αντίστοιχα ακολουθεί ο ψευδοκώδικας:

**ΒΗΜΑ 1: Αρχικοποίηση των πινάκων**

Εισάγονται τρεις πίνακες:

* M[i][j]: το βέλτιστο σκορ στοίχισης για τις πρώτες i θέσεις της A και j θέσεις της B, όπου η στοίχιση τελειώνει με κανονική αντιστοίχιση (όχι κενό)
* X[i][j]: το σκορ όταν η στοίχιση τελειώνει με κενό στην αλληλουχία B
* Y[i][j]: το σκορ όταν η στοίχιση τελειώνει με κενό στην αλληλουχία A

**ΒΗΜΑ 2: Αρχικοποίηση των πινάκων**

M[0][0] = 0

Για κάθε i = 1 .. n:

X[i][0] = –(ρ + (i–1)·σ), δηλαδή ξεκινά με κενό στη B

Για κάθε j = 1 .. m:

Y[0][j] = –(ρ + (j–1)·σ), δηλαδή ξεκινά με κενό στη A

Όλα τα υπόλοιπα κελιά των πινάκων αρχικοποιούνται με:

M[i][j], X[i][j], Y[i][j]=− ∞ (για i,j>0)

**ΒΗΜΑ 3: Υπολογισμός των κελιών**

Για κάθε i = 1 .. n και j = 1 .. m, υπολόγισε:

Συνάρτηση αντιστοίχισης:

S(I,j) =

Ενημέρωσε τους πίνακες:

M[i][j]=max

X[i][j] = max

Y[i][j] = max

**ΒΗΜΑ 4: Τελικό σκορ**

Το τελικό σκορ της στοίχισης είναι max( M[n][m], X[n][m], Y[n][m] ).

**ΒΗΜΑ 5: Ανακατασκευή στοίχισης**

Από το κελί με τη μέγιστη τιμή (ανάμεσα στα M[n][m], X[n][m], Y[n][m]), κάνουμε backtracking:

Επαναλαμβάνουμε:

* Αν ήρθαμε από διαγώνιο κελί → έγινε αντιστοίχιση χαρακτήρων
* Αν ήρθαμε από πάνω (δηλαδή από X[i−1][j]X[i-1][j]X[i−1][j]) → προστέθηκε κενό στη B
* Αν ήρθαμε από αριστερά (δηλαδή από Y[i][j−1]Y[i][j-1]Y[i][j−1]) → προστέθηκε κενό στην A

Συνεχίζουμε μέχρι να φτάσουμε στο M[0][0]M[0][0]M[0][0]

**ΒΗΜΑ 6: Επιστροφή στοίχισης και σκορ**

Η έξοδος περιλαμβάνει το τελικό σκορ στοίχισης και τις τελικές αλληλουχίες με τα κενά.

Η πολυπλοκότητα του αλγορίθμου καθολικής στοίχισης με συγγενική ποινή ασυμφωνίας (affine gap penalty) βασίζεται στην προσέγγιση του δυναμικού προγραμματισμού και διατηρείται πολυωνυμική. Ο αλγόριθμος συμπληρώνει τρεις πίνακες (M, X, Y) διαστάσεων n+1 × m+1, όπου:

* n = μήκος της πρώτης ακολουθίας A
* m = μήκος της δεύτερης ακολουθίας B

Σε κάθε κελί (i, j) γίνονται σταθερός αριθμός πράξεων (συγκρίσεις, max, άθροιση) ίσος με O(1), άρα η χρονική πολυπλοκότητα είναι ίση με O(n⋅m). Το γεγονός ότι υπάρχουν 3 πίνακες δεν αλλάζει την τάξη πολυπλοκότητας, είναι απλώς σταθερός πολλαπλασιαστής.

Ο αλγόριθμος αποθηκεύει τους 3 πίνακες των O(n·m) κελιών, άρα και η χωρική  πολυπλοκότητα θα είναι ίση με O(n⋅m).

Η χρήση affine gap penalties δεν αλλάζει την πολυπλοκότητα σε σχέση με απλή στοίχιση (π.χ. Needleman–Wunsch), αλλά απαιτεί περισσότερους πίνακες και πιο σύνθετη λογική, για να διαχειριστεί τη διαφορά μεταξύ "άνοιγμα κενού" και "επέκταση κενού".

Ο παρακάτω κώδικας σε Python υλοποιεί την καθολική στοίχιση δύο αλληλουχιών με affine gap penalty, δηλαδή με ποινή ανοίγματος κενού (ρ) και ποινή επέκτασης (σ). Χρησιμοποιεί τους τρεις πίνακες δυναμικού προγραμματισμού (M, X, Y) για να καταγράφει το βέλτιστο σκορ σε κάθε θέση, ανάλογα με το αν τελειώνει με match/mismatch ή κενό. Παράλληλα διατηρεί πίνακες αναδρομής (traceback) για να ξαναβρεί τη διαδρομή της στοίχισης. Τελικά, κάνει backtracking από το κελί (n, m) προς την αρχή, ανακατασκευάζοντας τις δύο στοιχισμένες ακολουθίες με τα κατάλληλα κενά. Επιστρέφει το τελικό σκορ και τη στοίχιση.

def **align\_with\_traceback**(A, B, rho, sigma):

    n = **len**(A)

    m = **len**(B)

*# Πίνακες δυναμικού προγραμματισμού*

    M = [[**float**('-inf')] \* (m + 1) for \_ in **range**(n + 1)]

    X = [[**float**('-inf')] \* (m + 1) for \_ in **range**(n + 1)]

    Y = [[**float**('-inf')] \* (m + 1) for \_ in **range**(n + 1)]

*# Πίνακες αναδρομής*

    traceback\_M = [[''] \* (m + 1) for \_ in **range**(n + 1)]

    traceback\_X = [[''] \* (m + 1) for \_ in **range**(n + 1)]

    traceback\_Y = [[''] \* (m + 1) for \_ in **range**(n + 1)]

*# Αρχικοποίηση*

    M[0][0] = 0

    traceback\_M[0][0] = 'START'

    for i in **range**(1, n + 1):

        X[i][0] = -rho - (i - 1) \* sigma

        traceback\_X[i][0] = 'X'

    for j in **range**(1, m + 1):

        Y[0][j] = -rho - (j - 1) \* sigma

        traceback\_Y[0][j] = 'Y'

*# Υπολογισμός πινάκων*

    for i in **range**(1, n + 1):

        for j in **range**(1, m + 1):

            s = 1 if A[i - 1] == B[j - 1] else -1

*# M[i][j]*

            options = [

                (M[i - 1][j - 1] + s, 'M'),

                (X[i - 1][j - 1] + s, 'X'),

                (Y[i - 1][j - 1] + s, 'Y')

            ]

            M[i][j], traceback\_M[i][j] = **max**(options)

*# X[i][j]*

            options = [

                (M[i - 1][j] - (rho + sigma), 'M'),

                (X[i - 1][j] - sigma, 'X')

            ]

            X[i][j], traceback\_X[i][j] = **max**(options)

*# Y[i][j]*

            options = [

                (M[i][j - 1] - (rho + sigma), 'M'),

                (Y[i][j - 1] - sigma, 'Y')

            ]

            Y[i][j], traceback\_Y[i][j] = **max**(options)

*# Τελικό σκορ και αρχική κατάσταση για backtracking*

    final\_score = **max**(M[n][m], X[n][m], Y[n][m])

    if final\_score == M[n][m]:

        matrix = 'M'

    elif final\_score == X[n][m]:

        matrix = 'X'

    else:

        matrix = 'Y'

*# Ανακατασκευή στοίχισης*

    aligned\_A, aligned\_B = "", ""

    i, j = n, m

    while i > 0 or j > 0:

        if matrix == 'M':

            prev = traceback\_M[i][j]

            i -= 1

            j -= 1

            aligned\_A = A[i] + aligned\_A

            aligned\_B = B[j] + aligned\_B

            matrix = prev

        elif matrix == 'X':

            prev = traceback\_X[i][j]

            i -= 1

            aligned\_A = A[i] + aligned\_A

            aligned\_B = '-' + aligned\_B

            matrix = prev

        elif matrix == 'Y':

            prev = traceback\_Y[i][j]

            j -= 1

            aligned\_A = '-' + aligned\_A

            aligned\_B = B[j] + aligned\_B

            matrix = prev

    return final\_score, aligned\_A, aligned\_B

Για ένα τυχαίο παράδειγμα που δοκιμάστηκε με τα εξής δεδομένα :

* A = "GATTACA"
* B = "GCATGCU"
* Gap opening penalty ρ = 2
* Gap extension penalty σ = 1,

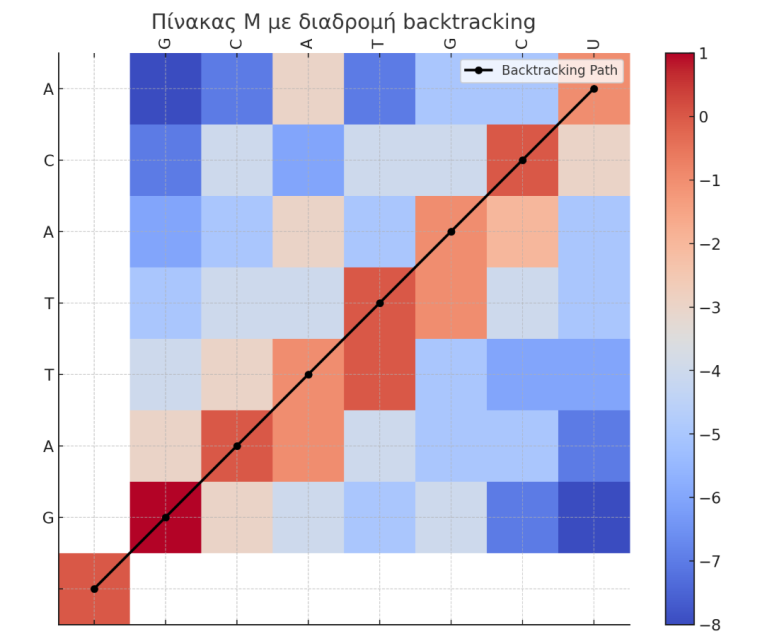
ο αλγόριθμος επέστρεψε το αποτέλεσμα :

**Τελικό Σκορ:** -1

**Στοίχιση A:** GATTACA

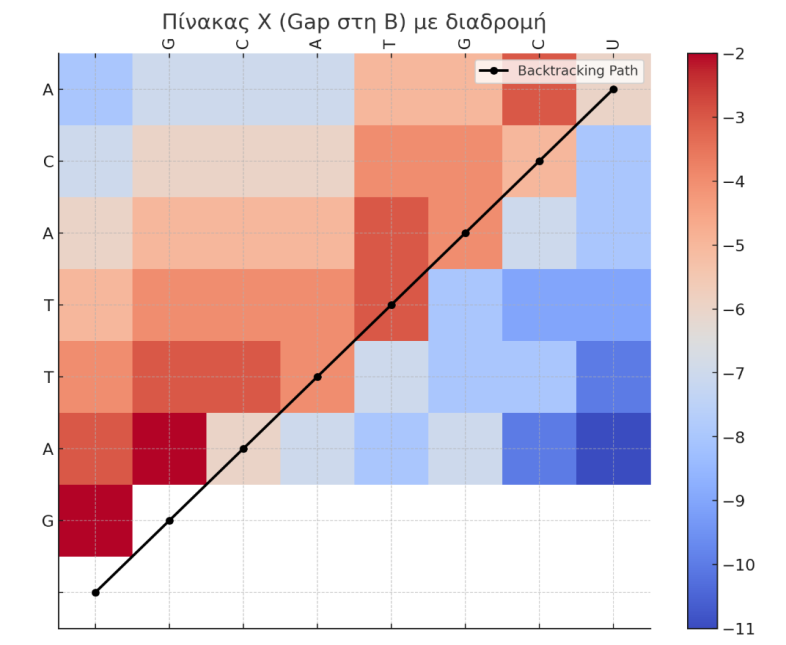
**Στοίχιση B:** GCATGCU

Το αποτέλεσμα επιβεβαιώνουν και τα παρακάτω heatmaps, τα οποία σχεδιάσθηκαν για την οπτική κατανόηση των αποτελεσμάτων.



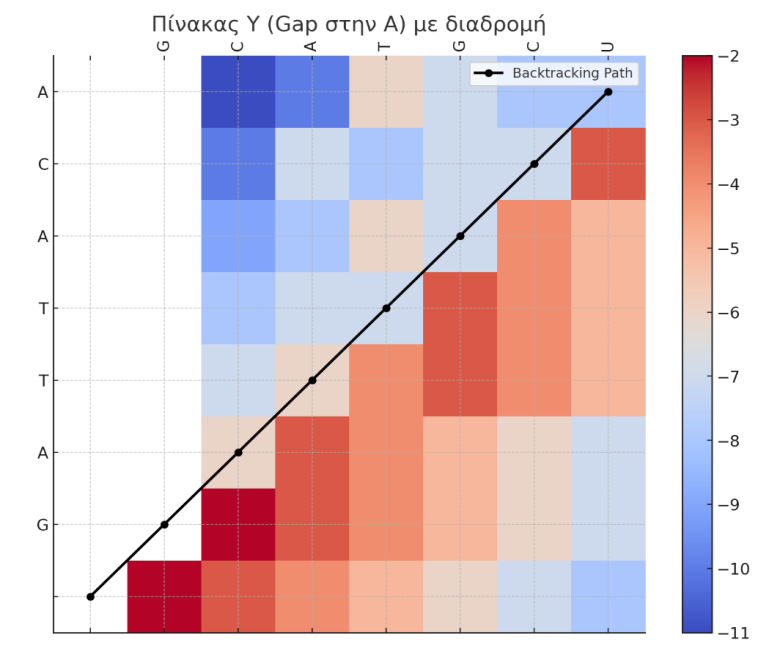
Heatmap M, που υπάρχει match/mismatch

Το heatmap του πίνακα M απεικονίζει τις περιοχές της στοίχισης, οι οποίες είναι πιο κοντά σε ικανοποιητικά σκορ. Το κόκκινο μονοπάτι προσεγγίζει την διαγώνιο, συνεπώς οι αλληλουχίες έχουν αρκετά ταιριάσματα σε παρόμοιες θέσεις. Επίσης, η τελική διαδρομή (backtracking) επιβεβαιώνει ότι δεν χρειάστηκαν πολλά gaps για να ευθυγραμμιστούν οι χαρακτήρες, δηλαδή δεν απαιτήθηκε από το σύστημα να παραληφθούν πολλά γράμματα για να εντοπιστούν ομοιότητες.



Heatmap Χ, πότε μπήκαν gaps στην Β

Όταν η στοίχιση τοποθετεί ένα γράμμα από την A απέναντι σε κενό (-) στην B, η διαδρομή περνάει από τον πίνακα X. Το heatmap του X απεικονίζει ποιες γραμμές (i) περιλαμβάνουν κενά στη B και ποιο είναι το κόστος τους (μπλε = χαμηλό σκορ). Έτσι, είναι κατανοητό σε ποιες περιπτώσεις απαιτήθηκε να προστεθεί κενό στην ακολουθία Β.



Heatmap Υ, πότε μπήκαν gaps στην A

Αντίστοιχα, όταν τοποθετείται γράμμα της B απέναντι σε κενό στην A, περνάει από τον πίνακα Y. Το heatmap του Y απεικονίζει ποιες στήλες (j) οδήγησαν σε κενά στην A και την επιρροή στο σκορ.

**Υποερώτημα (ii)**

Ο ψευδοκώδικας που παρουσιάζεται παρακάτω υλοποιεί έναν αποδοτικό αλγόριθμο για την εύρεση του μέγιστου κοινού προθέματος (Longest Common Prefix – LCP) για κάθε δυνατό ζεύγος συμβολοσειρών από ένα δοσμένο σύνολο k strings. Το βασικό ζήτημα του προβλήματος είναι ότι αν γινόταν σύγκριση κάθε ζεύγους με έναν απλό βρόχο, τότε θα κατέληγε σε πολυπλοκότητα O(k²·n), που καθίσταται απαγορευτική για μεγάλα δεδομένα. Για να ξεπεραστεί αυτό, ο αλγόριθμος βασίζεται στην κατασκευή ενός γενικευμένου δέντρου επιθεμάτων (generalized suffix trie), το οποίο επιτρέπει την ταυτόχρονη παρακολούθηση όλων των επιθεμάτων των strings σε κοινή δομή.

Το πρώτο βήμα του αλγορίθμου είναι η δημιουργία της δομής TrieNode, κάθε κόμβος της οποίας περιέχει ένα λεξικό children, που αντιστοιχεί στα επόμενα γράμματα, και ένα σύνολο indices, το οποίο κρατάει τους αριθμούς (ID) των strings που περνούν από τον εκάστοτε κόμβο. Στη συνέχεια, με τη χρήση της συνάρτησης InsertSuffix, ο αλγόριθμος εισάγει στο δέντρο όλα τα επιθέματα κάθε string. Για κάθε επίθεμα, δημιουργούνται νέοι κόμβοι (όπου χρειάζεται) και ενημερώνεται το αντίστοιχο indices ώστε να γνωρίζουμε από ποιες συμβολοσειρές προέρχεται το συγκεκριμένο μονοπάτι. Για να διαχωρίζονται σωστά τα strings μεταξύ τους, κάθε string ολοκληρώνεται με έναν μοναδικό χαρακτήρα (π.χ. $0, $1, κ.λπ.).

Αφού κατασκευαστεί το πλήρες generalized suffix trie, ο αλγόριθμος προχωρά με αναδρομική διάσχιση του δέντρου μέσω της συνάρτησης FindLCPs. Κατά τη διάσχιση, σε κάθε κόμβο, αν συμμετέχουν τουλάχιστον δύο strings (δηλαδή len(indices) ≥ 2), τότε το substring που φτάνει ως εκεί θεωρείται υποψήφιο μέγιστο κοινό πρόθεμα. Ο αλγόριθμος εξετάζει κάθε ζεύγος i < j από τα indices και ελέγχει αν το τρέχον πρόθεμα είναι μεγαλύτερο από το ήδη καταχωρημένο LCP για το συγκεκριμένο ζεύγος. Αν ναι, το ενημερώνει. Αυτό διασφαλίζει ότι στο τέλος θα έχουμε αποθηκεύσει το μέγιστο δυνατό κοινό πρόθεμα για κάθε σχετικό ζεύγος συμβολοσειρών.

Δομή TrieNode:

- children: λεξικό από χαρακτήρες σε TrieNode

- indices: σύνολο με τα ids των strings που περνούν από αυτόν τον κόμβο

Συνάρτηση InsertSuffix(root, string, string\_id):

Για i από 0 έως length(string) - 1:

node ← root

Για κάθε χαρακτήρα c στο επίθεμα string[i:]:

Αν c δεν υπάρχει στα children του node:

node.children[c] ← νέο TrieNode

node ← node.children[c]

node.indices ← node.indices ∪ {string\_id}

Συνάρτηση BuildGeneralizedSuffixTrie(strings):

root ← νέος TrieNode

Για κάθε string με δείκτη id στο strings:

InsertSuffix(root, string + "$" + id, id)

Επέστρεψε root

Συνάρτηση FindLCPs(node, prefix, αποτελέσματα):

Για κάθε χαρακτήρα c στα children του node:

child ← node.children[c]

Αν μέγεθος(child.indices) ≥ 2:

Για κάθε ζεύγος i, j με i < j από child.indices:

pair ← (i, j)

Αν pair δεν υπάρχει στα αποτελέσματα Ή

length(prefix + c) > length(αποτελέσματα[pair]):

αποτελέσματα[pair] ← prefix + c

FindLCPs(child, prefix + c, αποτελέσματα)

Συνάρτηση LongestCommonPrefixes(strings):

root ← BuildGeneralizedSuffixTrie(strings)

αποτελέσματα ← κενό λεξικό

FindLCPs(root, "", αποτελέσματα)

Επέστρεψε αποτελέσματα

Η πολυπλοκότητα του αλγορίθμου είναι εξαιρετικά σημαντική. Η κατασκευή του generalized suffix trie απαιτεί O(k·n) χρόνο, αφού για κάθε ένα από τα k strings εισάγονται περίπου n χαρακτήρες ανά επίθεμα. Επιπλέον, η αναδρομική εξαγωγή αποτελεσμάτων αφορά αποκλειστικά τα σημεία του δέντρου όπου υπάρχουν κοινά προθέματα (όχι όλους τους κόμβους), και το πλήθος των ζευγών για τα οποία πρέπει να καταγραφούν LCPs είναι α, που είναι και το μέγεθος της εξόδου και όχι επιπλέον κόστος. Έτσι, συνολικά έχουμε πολυπλοκότητα O(k·n + α), που είναι γραμμική ως προς την είσοδο και την έξοδο, γεγονός που καθιστά τον αλγόριθμο κατάλληλο για επεξεργασία πολύ μεγάλων συνόλων συμβολοσειρών.

Ο παρακάτω κώδικας σε Python κατασκευάζει ένα γενικευμένο δέντρο επιθεμάτων (suffix trie) για ένα σύνολο συμβολοσειρών. Για κάθε string, εισάγει όλα τα επιθέματά του στο δέντρο και σημειώνει ποια strings περνούν από κάθε κόμβο. Έπειτα, διατρέχει το δέντρο και εντοπίζει κόμβους που ανήκουν σε δύο ή περισσότερα strings. Για κάθε τέτοιο ζεύγος συμβολοσειρών, καταγράφει το μεγαλύτερο κοινό πρόθεμα που έχουν. Τελικά, επιστρέφει ένα λεξικό με το μέγιστο κοινό πρόθεμα για κάθε ζεύγος.

class **TrieNode**:

    def **\_\_init\_\_**(self):

        self.children = {}

        self.indices = **set**()

def **insert\_suffix**(trie, s, string\_id):

    for i in **range**(**len**(s)):

        node = trie

        for c in s[i:]:

            if c not in node.children:

                node.children[c] = **TrieNode**()

            node = node.children[c]

            node.indices.add(string\_id)

def **build\_generalized\_suffix\_trie**(strings):

    root = **TrieNode**()

    for idx, s in **enumerate**(strings):

**insert\_suffix**(root, s + f"${idx}", idx)

    return root

def **find\_longest\_common\_prefixes**(root, prefix='', results=None):

    if results is None:

        results = {}

    for char, child in root.children.items():

        if **len**(child.indices) >= 2:

            for i in child.indices:

                for j in child.indices:

                    if i < j:

                        pair = (i, j)

                        if pair not in results or **len**(prefix + char) > **len**(results[pair]):

                            results[pair] = prefix + char

**find\_longest\_common\_prefixes**(child, prefix + char, results)

    return results

Για ένα τυχαίο παράδειγμα που δοκιμάστηκε με τις λέξεις "banana", "bandana", "banish" και "banner" ο αλγόριθμος επέστρεψε το αποτέλεσμα :

Όπου στην αριστερή στήλη φαίνεται το ζεύγος που ελέγχεται και στην δεξιά στήλη φαίνεται το μέγιστο κοινό πρόθεμα των λέξεων. Οι κοινές ρίζες όπως "ban" και "ana$" εντοπίστηκαν σωστά και το $ στα προθέματα υποδεικνύει το μοναδικό τερματικό σύμβολο που προστέθηκε, ώστε να ξεχωρίζουμε τα strings στο δέντρο.

# **Ερώτημα 5ο**

[https://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Needleman-Wunsch#](https://rna.informatik.uni-freiburg.de/Teaching/index.jsp?toolName=Needleman-Wunsch)

**Υποερώτημα (i)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D |  | A | T | T | G | T | G | A | A | T | T |
|  | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 | -9 | -10 |
| T | -1 | -1 | 0 | -1 | -2 | -3 | -4 | -5 | -6 | -7 | -8 |
| T | -2 | -2 | 0 | 1 | 0 | -1 | -2 | -3 | -4 | -5 | -6 |
| A | -3 | -1 | -1 | 0 | 0 | -1 | -2 | -1 | -2 | -3 | -4 |
| G | -4 | -2 | -2 | -1 | 1 | 0 | 0 | -1 | -2 | -3 | -4 |
| T | -5 | -3 | -1 | -1 | 0 | 2 | 1 | 0 | -1 | -1 | -2 |
| T | -6 | -4 | -2 | 0 | -1 | 1 | 1 | 0 | -1 | 0 | 0 |
| A | -7 | -5 | -3 | -1 | -1 | 0 | 0 | 2 | 1 | 0 | -1 |
| A | -8 | -6 | -4 | -2 | -2 | -1 | -1 | 1 | 3 | 2 | 1 |
| G | -9 | -7 | -5 | -3 | -1 | -2 | 0 | 0 | 2 | 2 | 1 |
| T | -10 | -8 | -6 | -4 | -2 | 0 | -1 | -1 | 1 | 3 | 3 |
| G | -11 | -9 | -7 | -5 | -3 | -1 | 1 | 0 | 0 | 2 | 2 |

Η τιμή της βέλτιστης ολικής στοίχισης είναι **2** και η στοίχιση είναι:

\_TTAGTTAAGTG

ATT\_GTGAA\_TT

## **Υποερώτημα (ii)**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D |  | A | T | T | G | T | G | A | A | T | T |
|  | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| T | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| T | 0 | 0 | 1 | 2 | 1 | 1 | 0 | 0 | 0 | 1 | 2 |
| A | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 |
| G | 0 | 0 | 0 | 0 | 2 | 1 | 1 | 0 | 0 | 0 | 0 |
| T | 0 | 0 | 1 | 1 | 1 | 3 | 2 | 1 | 0 | 1 | 1 |
| T | 0 | 0 | 1 | 2 | 1 | 2 | 2 | 1 | 0 | 1 | 2 |
| A | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 3 | 2 | 1 | 1 |
| A | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 2 | 4 | 3 | 2 |
| G | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 3 | 3 | 2 |
| T | 0 | 0 | 1 | 1 | 0 | 2 | 1 | 0 | 2 | 4 | 4 |
| G | 0 | 0 | 0 | 0 | 2 | 1 | 3 | 2 | 1 | 3 | 3 |

Η τιμή της βέλτιστης τοπικής στοίχισης είναι **4** και η στοίχιση είναι:

TTAGTTAA

TT\_GTGAA

# **Ερώτημα 6ο**

Για τα δύο υποερωτήματα επιλέγεται η ανάπτυξη σε Jupyter Notebook, στο περιβάλλον Google Colab.

## **Υποερώτημα (i)**

Για το πρώτο ερώτημα επιλέγεται η πρωτεΐνη με UniProt ID P08246. Αναπτύσσεται κώδικας σε python (χρησιμοποιείται biopython).

Αρχικά γίνεται request από το url η πρωτεΐνη, με βάση το id της και αποθηκεύεται σε .fasta αρχείο.

# protein download - FASTA format

uniprot\_id = "P08246"

url = f"https://www.uniprot.org/uniprot/{uniprot\_id}.fasta"

response = requests.get(url)

with open("P08246.fasta", "w") as f:

f.write(response.text)

Στην συνέχεια, γίνεται χρήση του εργαλείου BLASTP. Το συγκεκριμένο εργαλείο δέχεται ως είσοδο την πρωτεΐνη P08246 και αναζητά για όμοιες ακολουθίες σε άλλες πρωτεΐνες, σε μία μεγάλη βάση δεδομένων. Το εργαλείο BLASTP παραμετροποιείται για να αποκλείσει από οποιαδήποτε επεξεργασία τις ανθρώπινες πρωτεΐνες (entrez\_query="NOT Homo sapiens[Organism]"). Τα αποτελέσματα των ευθυγραμμίσεων καταγράφονται σε ένα .txt αρχείο.

record = SeqIO.read("P08246.fasta", format="fasta")

# non human

result\_handle = NCBIWWW.qblast("blastp", "nr", record.seq, entrez\_query="NOT Homo sapiens[Organism]", hitlist\_size=10)

with open("blast\_result.xml", "w") as f:

f.write(result\_handle.read())

Κατόπιν, γίνεται ανάκτηση των ομόλογων ακολουθιών, που προέκυψαν από τα αποτελέσματα του BLASTP και εξάγονται οι πρώτες οκτώ ομόλογες πρωτεϊνικές ακολουθίες.

result\_handle = open("blast\_result.xml")

blast\_record = NCBIXML.read(result\_handle)

sequences = []

for i, alignment in enumerate(blast\_record.alignments[:8]):

Για κάθε μία ευθυγραμμισμένη ακολουθία δημιουργεί ένα αντικείμενο SeqRecord που συνοδεύεται από ένα μοναδικό αναγνωριστικό. Έπειτα αποθηκεύει όλες τις ομόλογες ακολουθίες σε αρχείο .FASTA για να αξιοποιηθούν στην πολλαπλή στοίχιση.

sequences = []

for i, alignment in enumerate(blast\_record.alignments[:8]):

hsp = alignment.hsps[0]

seq = Seq(hsp.sbjct.replace("-", ""))

record = SeqRecord(seq=seq, id=f"Homolog\_{i+1}", description=alignment.hit\_def)

sequences.append(record)

SeqIO.write(sequences, "homologs.fasta", "fasta")

Στην συνέχεια με την εντολή ClustalOmegaCommandline της Biopython γίνεται **πολλαπλή στοίχιση** (multiple sequence alignment) στις ομόλογες πρωτεϊνικές ακολουθίες. Η παράμετρος verbose=True επιτρέπει την εκτύπωση πληροφοριών της εκτέλεση και η auto=True επιτρέπει την εκτέλεση της εντολής με αυτόματες παραμέτρους. Επιπλέον ορίζεται η έξοδος σε αρχείο .aln.

from Bio.Align.Applications import ClustalOmegaCommandline

clustalomega\_cline = ClustalOmegaCommandline(infile="homologs.fasta", outfile="multiple.aln", verbose=True, auto=True, outfmt="clu")

clustalomega\_cline()

Τέλος, για την στοίχιση κατά ζεύγη επιλέγεται global στοίχιση (τύπου Needleman-Wunsch)

για τις πρώτες δύο ακολουθίες της λίστας sequence Το global σημαίνει στοίχιση καθ' όλο το μήκος της ακολουθίας και το xx σημαίνει ότι δεν υπάρχει ποινή και συνεπώς ισχύει το score +1 για ταύτιση και 0 για mismatch ή κενό. Επαναλαμβάνεται για όλες τις ακολουθίες.

from Bio import pairwise2

from Bio.pairwise2 import format\_alignment

alignments = pairwise2.align.globalxx(sequences[0].seq, sequences[1].seq)

with open("pairwise.aln", "w") as f:

for aln in alignments:

f.write(format\_alignment(\*aln))

Τα αποτελέσματα της εκτέλεσης του κώδικα παρατίθενται στο .zip αρχείο. Δεν προστέθηκαν στην παρούσα αναφορά, για να μην αλλοιωθεί η απεικόνιση τους (δεν γινόταν σωστή στοίχιση)

## **Υποερώτημα (ii)**

### **BLASTP**

Για να βρεθεί σε ποιον οργανισμό ανήκει η ακολουθία της πρωτεΐνης γίνεται αναζήτηση της ακολουθίας στο BLAST.

Αρχικά ορίζεται η ακολουθία και το αντικείμενο SeqRecord για να υποβληθεί η ακολουθία στο BLAST σε .fasta μορφή.

sequence = ("GATGAPGIAGAPGFPGARGAPGPQGPSGAPGPKXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXGVQGPPGPQGPRXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXGSAGPPGATGFPGAAGRXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXGVVGLPGQR")

record = SeqRecord(Seq(sequence), id="sequence", description="Unknown protein")

Εκτελείται το BLASTP στο NCBI, όπου ορίζονται οι παράμετροι:

* blastp εργαλείο για πρωτεΐνες
* nr non-redundant database
* record.format("fasta")μετατρέπει το αντικείμενο σε .fasta

και φορτώνεται το πρώτο αποτέλεσμα που επέστρεψε ο NCBI server.

# BLASTP - NCBI

result\_handle = NCBIWWW.qblast("blastp", "nr", record.format("fasta"))

# analyzing results

blast\_record = NCBIXML.read(result\_handle)

Στην συνέχεια γίνεται επιλογή των αποτελεσμάτων που προέρχονται μόνο από οργανισμούς των οποίων το όνομα ξεκινάει με "Tyr". Δημιουργείται μία άδεια λίστα tyr\_hits, για να αποθηκευτούν τα αποτελέσματα που πληρούν τα κριτήρια της αναζήτησης.

tyr\_hits = []

Διατρέχεται κάθε αποτέλεσμα της ευθυγράμμισης από την απάντηση του BLAST και αποθηκεύεται.

for alignment in blast\_record.alignments:

    hit\_def = alignment.hit\_def

Έπειτα γίνεται αναζήτηση του ονόματος του οργανισμού, το οποίο υπάρχει στο hit\_def και αν εντοπιστεί οργανισμός αποθηκεύει το όνομα του οργανισμού.

    organism\_match = re.search(r'\[(.\*?)\]', hit\_def)

    if organism\_match:

        organism = organism\_match.group(1)

Στην συνέχεια γίνεται έλεγχος εάν το όνομα του οργανισμού ξεκινάει με “Tyr” και εξάγει το accession ID. Αν το όνομα ξεκινάει με “Tyr” το αποθηκεύει στην λίστα.

Ο έλεγχος σταματάει εάν έχουν βρεθεί 8 αποτελέσματα.

        if re.match(r'Tyr\w\*', organism, re.IGNORECASE):

            accession = alignment.accession

            tyr\_hits.append((accession, hit\_def, organism))

            if len(tyr\_hits) == 8:

                break

Τέλος, εκτυπώνονται τα αποτελέσματα της αναζήτησης.

# Print results

print("\nTop BLASTP hits from organisms starting with 'Tyr':\n")

for i, (acc, hit\_def, organism) in enumerate(tyr\_hits, 1):

    print(f"{i}. {acc} — {hit\_def}")

    print(f"   Organism: {organism}")

    print()

### **Σχολιασμός Αποτελεσμάτων**

Ο παραπάνω κώδικας επιστρέφει ως αποτελέσματα τα ακόλουθα:

Top BLASTP hits from organisms starting with 'Tyr':

1. RecName: Full=Collagen alpha-1(I) chain; AltName: Full=Alpha-1 type I collagen [Tyrannosaurus rex] — RecName: Full=Collagen alpha-1(I) chain; AltName: Full=Alpha-1 type I collagen [Tyrannosaurus rex]

Organism: Tyrannosaurus rex

Η ανακάλυψη της πρωτεΐνης κολλαγόνου τύπου I, αλυσίδας α1 (κωδικός P0C2W2.2) στον Tyrannosaurus rex αποτέλεσε ορόσημο στη μοριακή παλαιοντολογία. Η μελέτη απολιθώματος ηλικίας 68 εκατομμυρίων ετών (Asara κ.ά., 2007), υπέδειξε την εξελικτική συγγένεια των δεινοσαύρων με τα πτηνά . Ωστόσο, η ερμηνεία αυτή αμφισβητήθηκε. Η συγκεκριμένη έρευνα προκάλεσε έντονες αντιδράσεις και επανάσταση στον επιστημονικό χώρο, με αρκετούς να την αμφισβητούν (Pevzner κ.ά., 2008) και οδήγησε σε περαιτέρω μελέτες.

Οι επιστήμονες προσπάθησαν να αξιολογήσουν τις συνθήκες διατήρησης του κολλαγόνου (Buckley κ.ά., 2008), επισημαίνοντας ότι η θερμοκρασία και η περιβαλλοντική σταθερότητα είναι κρίσιμοι παράγοντες για τη μακροχρόνια διατήρηση των πρωτεϊνών . Αν και η παρουσία κολλαγόνου σε τόσο αρχαία δείγματα παραμένει αμφιλεγόμενη, η ανακάλυψη αυτή άνοιξε νέους δρόμους στην κατανόηση της εξελικτικής σχέσης μεταξύ δεινοσαύρων και πτηνών, ενισχύοντας την υπόθεση της στενής συγγένειάς τους.

# **Βιβλιογραφία**

[Desai, D., Kantliwala, S. V., Vybhavi, J., Ravi, R., Patel, H., & Patel, J. (2024). Review of AlphaFold 3: Transformative Advances in Drug Design and Therapeutics. *Cureus*. https://doi.org/10.7759/cureus.63646](https://www.zotero.org/google-docs/?ZiBFMm)

Asara, J. M., Schweitzer, M. H., Freimark, L. M., Phillips, M., & Cantley, L. C. (2007). Protein sequences from mastodon and Tyrannosaurus rex revealed by mass spectrometry. *Science (New York, N.Y.)*, *316*(5822), 280–285. https://doi.org/10.1126/science.1137614

Bonvin, A. (2025). *Homology modelling of the mouse MDM2 protein*. Bonvin Lab. https://www.bonvinlab.org/education/molmod\_online/modelling/

Buckley, M., Walker, A., Ho, S. Y. W., Yang, Y., Smith, C., Ashton, P., Oates, J. T., Cappellini, E., Koon, H., Penkman, K., Elsworth, B., Ashford, D., Solazzo, C., Andrews, P., Strahler, J., Shapiro, B., Ostrom, P., Gandhi, H., Miller, W., … Collins, M. J. (2008). Comment on ‘Protein sequences from mastodon and Tyrannosaurus rex revealed by mass spectrometry’. *Science (New York, N.Y.)*, *319*(5859), 33; author reply 33. https://doi.org/10.1126/science.1147046

*Help | SWISS-MODEL*. (χ.χ.). Ανακτήθηκε 24 Απρίλιος 2025, από https://swissmodel.expasy.org/docs/help

Pevzner, P. A., Kim, S., & Ng, J. (2008). Comment on ‘Protein sequences from mastodon and Tyrannosaurus rex revealed by mass spectrometry’. *Science (New York, N.Y.)*, *321*(5892), 1040; author reply 1040. https://doi.org/10.1126/science.1155006

Studer, G., Rempfer, C., Waterhouse, A. M., Gumienny, R., Haas, J., & Schwede, T. (2020). QMEANDisCo-distance constraints applied on model quality estimation. *Bioinformatics (Oxford, England)*, *36*(6), 1765–1771. https://doi.org/10.1093/bioinformatics/btz828

[Edgar, R. C., & Sjölander, K. (2004). A comparison of scoring functions for protein sequence profile alignment. *Bioinformatics*, *20*(8), 1301–1308. https://doi.org/10.1093/bioinformatics/bth090](https://www.zotero.org/google-docs/?ZiBFMm)

[Manfredi, M., Vazzana, G., Savojardo, C., Martelli, P. L., & Casadio, R. (2025). AlphaFold2 and ESMFold: A large-scale pairwise model comparison of human enzymes upon Pfam functional annotation. *Computational and Structural Biotechnology Journal*, *27*, 461–466. https://doi.org/10.1016/j.csbj.2025.01.008](https://www.zotero.org/google-docs/?ZiBFMm)

[Stampelou, M., Ladds, G., & Kolocouris, A. (2024). Computational Workflow for Refining AlphaFold Models in Drug Design Using Kinetic and Thermodynamic Binding Calculations: A Case Study for the Unresolved Inactive Human Adenosine A3 Receptor. *The Journal of Physical Chemistry B*, *128*(4), 914–936. https://doi.org/10.1021/acs.jpcb.3c05986](https://www.zotero.org/google-docs/?ZiBFMm)