

**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ: ΒΙΟΠΛΗΡΟΦΟΡΙΚΗ**

Επί των φοιτητών: Μόσχου Δημήτριου - Π18209

Σιάτρα Απόστολου – Π18215

Διδάσκων: Πικράκης Άγγελος

Πειραιάς, 2021



**ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ**

**ΕΡΓΑΣΙΑ ΣΤΟ ΜΑΘΗΜΑ: ΒΙΟΠΛΗΡΟΦΟΡΙΚΗ**

***Τελική Εργασία Μαθήματος***

Επί των φοιτητών: Μόσχου Δημήτριου - Π18209

Σιάτρα Απόστολου – Π18215

Διδάσκων: Πικράκης Άγγελος

Πειραιάς, 2021

Πίνακας περιεχομένων

**1. ΘΕΜΑ 1ο**

2. **ΘΕΜΑ 2ο**

3. **ΘΕΜΑ 3ο**

**4. ΘΕΜΑ 4ο**

**4.1 Ερώτημα 1ο**

**4.2 Ερώτημα 2ο**

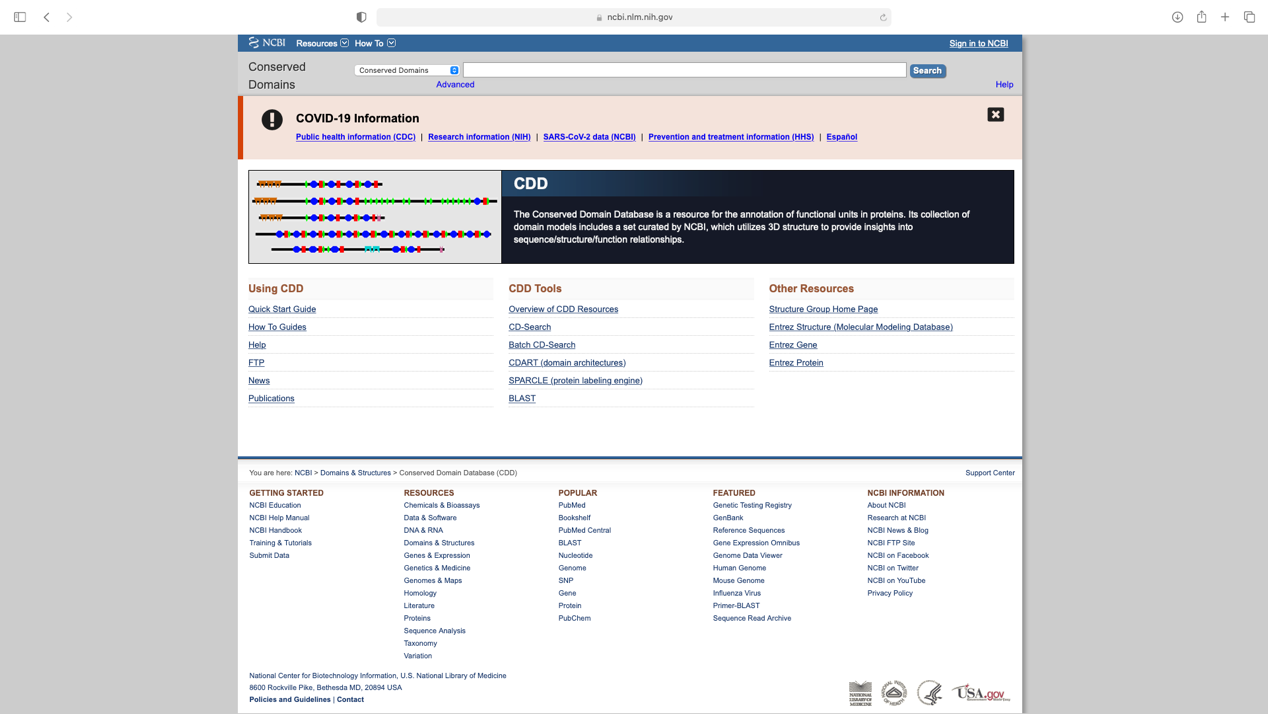
**4.3 Ερώτημα 3ο**

**4.4 Ερώτημα 4ο**

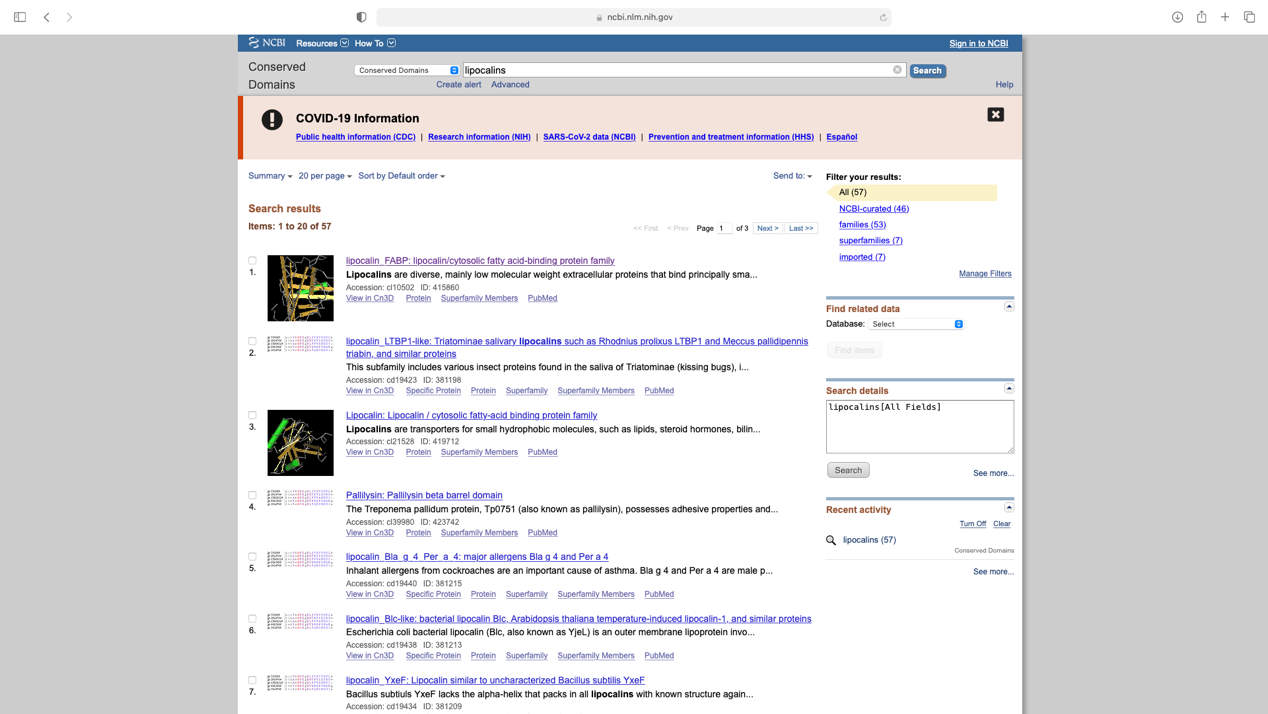
**4.5 Ερώτημα 5ο**

**1. ΘΕΜΑ 1ο**

Αρχικώς, μεταβαίνουμε στη βάση δεδομένων των συντηρημένων δοκιμών επικρατειών στο NCBI



Στη συνέχεια, εισάγουμε τον όρο λιποκαλίνες (lipocalins), ή κάποιο άλλο όνομα οικογένειας της επιλογής μας.



Έπειτα, αφότου κάνουμε την επιλογή μας, θα επιλέξουμε το mFasta ως μορφή αρχείου και στην συνέχεια θα εκτελέσουμε το Reformat. Το αποτέλεσμα που θα εμφανιστεί θα είναι μία πολλαπλή στοίχιση αλληλουχιών.

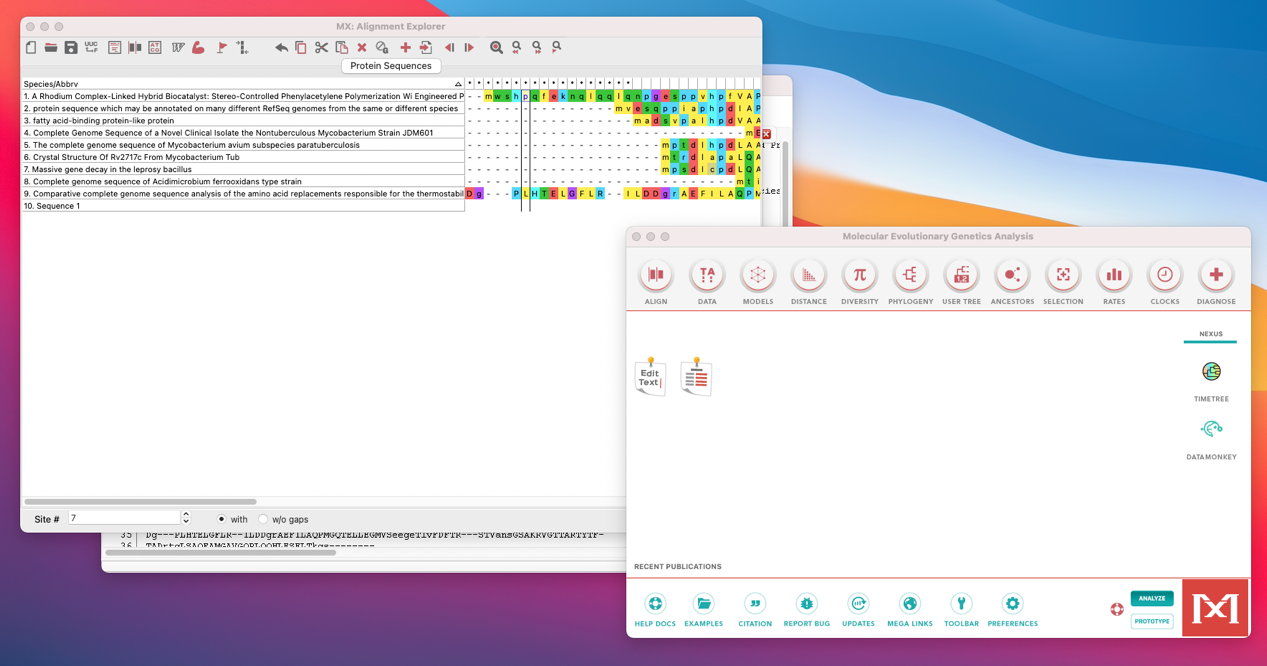
Εικόνα που περιέχει κείμενο, εφημερίδα, στιγμιότυπο οθόνης, έγγραφο

Περιγραφή που δημιουργήθηκε αυτόματα

Μετά, θα αντιγράψουμε το αποτέλεσμα αυτό σε έναν text editor της επιλογής μας (στη συγκεκριμένη περίπτωση χρησιμοποιήθηκε το VS Code και το TextEdit της Apple), έτσι ώστε να απλοποιήσουμε τα ονόματα των αλληλουχιών.Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

Εισάγουμε το αρχείο .txt στο πρόγραμμα MEGA (στην προκειμένη περίπτωση χρησιμοποιείται η έκδοση Χ ή αλλιώς 10), θα στοιχίσουμε τις αλληλουχίες και θα τις αποθηκεύσουμε σε αρχεία τύπου .mas.



Ακόμη, θα επιλέξουμε τη διαδικασία Phylogeny έτσι ώστε να δημιουργήσουμε δέντρα με τις μεθόδους ένωσης γειτόνων, μέγιστης πιθανοφάνειας ή άλλων.

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

**2. ΘΕΜΑ 2ο**

Παρακάτω παρατίθεται ο κώδικας του συγκεκριμένου θέματος:

**from** **numpy** **import** log **as** ln

dual\_states=('A','B') #Δυο καταστασεις (Α,Β)

possibility\_a = [['A','G','T','C'],[**0.4**,**0.4**,**0.1**,**0.1**]] #Η πιθανοτητα να εκπεμψει πουρινες και πυριμιδινες στην κατασταση A

possibility\_b=[['T','C','A','G'],[**0.3**,**0.3**,**0.2**,**0.2**]] #Η πιθανοτητα να εκπεμψει πουρινες και πυριμιδινες στην κατασταση Β

goal=('G','G','C','T') #Η ζητουμενη αλληλουχια

pos\_a\_to\_a\_and\_b=[**0.9**,**0.1**] #Η πιθανοτητα να συνεχισει απο την Α κατασταση στην Α και στην Β

pos\_b\_to\_a\_and\_b=[**0.1**,**0.9**] #Η πιθανοτητα να συνεχισει απο την Β κατασταση στην Β και στην Α

initial\_pos=[**0.5**,**0.5**] ##Η αρχικη πιθανοτητα να ξεκινησει απο το Α και το Β

**def** **viterbi\_algo**(possibility\_a,pos\_a\_to\_a\_and\_b,goal,initial\_pos,possibility\_b,from\_b\_to\_b\_and\_a,dual\_states):

temp\_a=[] #Αποθηκευονται οι τιμες της Α καταστασης απο τον viterbi αλγοριθμο

temp\_b=[] #Αποθηκευονται οι τιμες της Β καταστασης απο τον viterbi αλγοριθμο

flag=**True**

temp1=**False**

temp2=**False**

start=**False**

log\_num=**0**

i=**0**

best\_path=[]

**while**(flag): #Οσο το flag παραμενει true η while θα τρεχει

**for** j **in** range(**0**,len(goal)): #Μπαινει σε μια for με μεγεθος οση ειναι η ζητουμενη αλληλουχια(goal)

**if**(start==**False**): #Εαν το start ειναι false σημαινει οτι βρισκομαστε στην πρωτη κατασταση με ζητουμενο το G

**if**(possibility\_a[**0**][j]==goal[i]): #Εαν το possibility\_a[0][j] ισουται με το ζητουμενο

temp\_a.append(initial\_pos[**0**]\*possibility\_a[**1**][j]) #τοτε η temp\_a παιρνει την αρχικη πιθανοτητα του A και την πολλαπλασιαζει με την πιθανοτητα του ζητουμενου νουκλεοτιδιου

temp1=**True**

**if**(possibility\_b[**0**][j]==goal[i]):

temp\_b.append(initial\_pos[**1**]\*possibility\_b[**1**][j]) ##τοτε η temp\_b παιρνει την αρχικη πιθανοτητα του Β και την πολλαπλασιαζει με την πιθανοτητα του ζητουμενου νουκλεοτιδιου

temp2=**True**

**if**(temp1 **and** temp2): #Εαν το temp1 και το temp2 ειναι true τοτε εχουμε βρει το πρωτο ζητουμενο(goal[0])

start=**True** #Το start γινεται True οποτε το προγραμμα δεν μπαινει ξανα στην αρχικη if

i+=**1**

**if**(temp\_a[**0**]>temp\_b[**0**]): #Εαν το temp\_a που βρηκαμε ειναι μεγαλυτερο απο το temp\_b που βρηκαμε τοτε

best\_path.append(dual\_states[**0**]) #Το best\_path παιρνει την πρωτη κατασταση

log\_num+= ln(temp\_a[**0**]) #Μετατρεπει τις αριθμιτικες τιμες σε λογαριθμικες και τις προσθετει σε μια μεταβλητη log\_num

**elif**(temp\_a[**0**]<temp\_b[**0**]): #Αλλιως το best\_path παιρνει την δευτερη κατασταση

best\_path.append(dual\_states[**1**])

log\_num+= ln(temp\_b[**0**]) #Μετατρεπει τις αριθμιτικες τιμες σε λογαριθμικες και τις προσθετει σε μια μεταβλητη log\_num

**break** #Αφου βρει ποιο ειναι μεγαλυτερο κανει break απο την for και ξεκιναει για τις υπολοιπες καταστασεις αφου το start γινεται True στην σειρα 36

**if**(start **and** temp1 **and** temp2): #Εαν και τα τρια flag ειναι true ψαχνουμε τις υπολοιπες καταστασεις

temp1=**False** #Κανουμε το temp1 και το temp2 False ωστε να τα χρησιμοποιησουμε ξανα στον υπολοιπο κωδικα

temp2=**False**

**if**(start==**True**): #Εαν το start ειναι True

**if**(possibility\_a[**0**][j]==goal[i] **and** best\_path[i-**1**]=='A'): #Εαν το νουκλεοτιδιο της possibility\_a ισουται με το ζητουμενο νουκλεοτιδιο και το best\_path ισουται με την πρωτη κατασταση(Α)

temp\_a.append(round(pos\_a\_to\_a\_and\_b[**0**]\*possibility\_a[**1**][j]\*temp\_a[i-**1**],**10**)) #Η temp\_a παιρνει την πιθανοτητα να παει απο το Α στο Α πολλαπλασιαζοντας την με την πιθανοτητα

#του ζητουμενου νουκλεοτιδιου και το temp\_a της προηγουμενης καταστασης

temp1=**True**

**elif**(possibility\_a[**0**][j]==goal[i] **and** best\_path[i-**1**]=='B'): #Αλλιως εαν το νουκλεοτιδιο της possibility\_a ισουται με το ζητουμενο νουκλεοτιδιο και το best\_path ισουται με την δευτερη κατασταση(Β)

temp\_a.append(round(pos\_b\_to\_a\_and\_b[**0**]\*possibility\_a[**1**][j]\*temp\_b[i-**1**],**10**)) #Η temp\_a παιρνει την πιθανοτητα να παει απο το Β στο Α πολλαπλασιαζοντας την με την πιθανοτητα

#του ζητουμενου νουκλεοτιδιου και το temp\_b της προηγουμενης καταστασης

temp1=**True**

**if**(possibility\_b[**0**][j]==goal[i] **and** best\_path[i-**1**]=='A'): #Εαν το νουκλεοτιδιο της possibility\_b ισουται με το ζητουμενο νουκλεοτιδιο και το best\_path ισουται με την πρωτη κατασταση(Α)

temp\_b.append(round(pos\_a\_to\_a\_and\_b[**1**]\*possibility\_b[**1**][j]\*temp\_a[i-**1**],**10**)) #Η temp\_b παιρνει την πιθανοτητα να παει απο το Α στο Β πολλαπλασιαζοντας την με την πιθανοτητα

#του ζητουμενου νουκλεοτιδιου και το temp\_a της προηγουμενης καταστασης

temp2=**True**

**elif**(possibility\_b[**0**][j]==goal[i] **and** best\_path[i-**1**]=='B'): #Αλλιως εαν το νουκλεοτιδιο της possibility\_b ισουται με το ζητουμενο νουκλεοτιδιο και το best\_path ισουται με την δευτερη κατασταση(Β)

temp\_b.append(round(pos\_b\_to\_a\_and\_b[**1**]\*possibility\_b[**1**][j]\*temp\_b[i-**1**],**10**)) #Η temp\_b παιρνει την πιθανοτητα να παει απο το Β στο Β πολλαπλασιαζοντας την με την πιθανοτητα

#του ζητουμενου νουκλεοτιδιου και το temp\_b της προηγουμενης καταστασης

temp2=**True**

**if**(temp1==**True** **and** temp2==**True**): #Εαν το temp1 και το temp2 ειναι True

**if**(temp\_a[i]>temp\_b[i]): #Εαν το temp\_a ειναι μεγαλυτερο του temp\_b

best\_path.append(dual\_states[**0**]) #Τοτε το best\_path παιρνει την πρωτη κατασταση

log\_num+= ln(temp\_a[i]) #Μετατρεπει τις αριθμιτικες τιμες σε λογαριθμικες και τις προσθετει σε μια μεταβλητη log\_num

**else**:

best\_path.append(dual\_states[**1**]) #Αλλιως το best\_path παιρνει την δευτερη κατασταση

log\_num+= ln(temp\_b[i]) #Μετατρεπει τις αριθμιτικες τιμες σε λογαριθμικες και τις προσθετει σε μια μεταβλητη log\_num

i+=**1**

**if**(i==**4** **and** len(best\_path)==**4**): #Εαν το i εχει γινει 4 και το best\_path εχει 4 τιμες τοτε

flag=**False** #Το flag γινεται false και οταν τελειωσει η while βγαινει απο αυτην

**for** i **in** range(**0**,len(best\_path)): #Με μια for εκτυπωνει τις τιμες της Α και της Β καταστασης καθως και το μονοπατι με την μεγαλυτερη πιθανοτητα

print('A is: ',temp\_a[i],'and B is: ',temp\_b[i])

print('The best path is: ',best\_path,'and the rounded sum of the logarithms for best\_path is: ',round(log\_num,**4**)) #Εκτυπωνει το καλυτερο μονοπατι και το αθροισμα των λογαριθμικο αθροισμα

**break** #Αφου εχουν εκτπωθει ολα κανει break, βγαινει απο την for και αφου το flag εχει γινει False τελειωνει και η while

viterbi\_algo(possibility\_a,pos\_a\_to\_a\_and\_b,goal,initial\_pos,possibility\_b,pos\_b\_to\_a\_and\_b,dual\_states) #Καλειται η viterbi\_algo

Παράδειγμα εκτέλεσης:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

*Επεξήγηση του συγκεκριμένου θέματος:*

Σκοπός της εργασίας είναι η δημιουργία ενός προγράμματος το οποίο έχει ως στόχο να εκτελεί τον αλγόριθμο viterbi και να βρίσκει την πιθανότερη διαδρομή που θα ακολουθήσουν οι δυο καταστάσεις (α, β) για την αλληλουχία GGCT. Το πρόγραμμα παίρνει ως αρχικό όρισμα την πιθανότητα να εκπέμψει πουρίνες και πυριμιδίνες στις δύο καταστάσεις(α, β), την πιθανότητα να συνεχίσει από την κατάσταση (α) στην κατάσταση (β) και αντίστροφα, την αρχική πιθανότητα και την ζητούμενη αλληλουχία. Στην συνέχεια εκτελεί τον αλγόριθμο viterbi και εκτυπώνει το βέλτιστο μονοπάτι.

3. ΘΕΜΑ 3ο

Παρακάτω παρατίθεται ο κώδικας του συγκεκριμένου θέματος:

**import** **random**

**from** **Bio** **import** SeqIO

#Φορτώνουμε τις 2 αλληλουχιες

**for** seq1 **in** SeqIO.parse("liver.fasta", "fasta"):

chromo\_liver = seq1.seq

**for** seq2 **in** SeqIO.parse("brain.fasta", "fasta"):

chromo\_brain = seq2.seq

chromo\_liver = len(seq1) #Βρίσκουμε το μέγεθος της πρώτης αλληλουχιάς

chromo\_brain = len(seq2) #Βρίσκουμε το μέγεθος της δεύτερης αλληλουχιάς

player = **1**

#Εδώ ελέγχουμε αν κέρδισε καποιος παίχτης

**def** **check**(player):

**if**(chromo\_brain==**0** **or** chromo\_liver==**0**):

print("The winner is Player ",player)

**while**(**True**): #Το παιχνίδι παίζει μέχρι να νικήσει ένας από τους δύο παίχτες

**if**(chromo\_brain!=**0** **and** chromo\_liver!=**0**): #Αν δεν εχει μηδενισει καμια αλληλουχια μπαινει στην if

print("The player number ",player ," plays now. **\n**Choose if you would like to erase from the 1.first(liver), 2.second(brain) or 3.from both sequences. "+

"**\n**Type 1, 2, or 3: ")

**if**(chromo\_brain<chromo\_liver): #αν η αλληλουχια 1 ειναι μικροτερη απο την 2 τοτε ο τυχαιος αριθμος θα ειναι απο το 1 μεχρι το μεγεθος της αλληλουχιας 1

smaller\_seq=chromo\_brain

**else**:

smaller\_seq=chromo\_liver #αν η αλληλουχια 2 ειναι μικροτερη απο την 1 τοτε ο τυχαιος αριθμος θα ειναι απο το 1 μεχρι το μεγεθος της αλληλουχιας 2

randomness = random.randint(**1**,smaller\_seq) #τυχαιος αρθμος απο το 1 μεχρι τον αριθμο που ορισαμε απο πανω

choice= int(input()) #Ο χρηστης δινει στο προγραμμα το τι θελει να κανει

**while**(**True**): #Τρέχει μεχρι να δεχτει σωστο τυπο input απο τον χρηστη

**if**(choice==**1** **or** choice==**2** **or** choice==**3**): #Αν η επιλογη ειναι 1 η 2 η 3 βγαινει απο την while και συνεχιζει

**break**

**else**:

print("Available choices: 1, 2 or 3 !!") #Αν η επιλογη δεν ειναι 1 η 2 η 3 εκτυπωνονται οι διαθεσιμες επιλογες

choice = int(input()) #Ο χρηστης δινει στο προγραμμα το τι θελει να κανει

**if**(choice==**1**): #Αν επιλεξει την επιλογη 1

chromo\_liver-=randomness #Αφαιρουμε απο την αλληλουχια 1 τον τυχαιο αριθμο randomness

check(player) #Ελεγχουμε μηπως τελειωσε το παιχνιδι

**elif**(choice==**2**):#Αν επιλεξει την επιλογη 2

chromo\_brain-=randomness #Αφαιρουμε απο την αλληλουχια 2 τον τυχαιο αριθμο randomness

check(player) #Ελεγχουμε μηπως τελειωσε το παιχνιδι

**elif**(choice==**3**):#Αν επιλεξει την επιλογη 3

chromo\_brain-=randomness #Αφαιρουμε απο την αλληλουχια 1 τον τυχαιο αριθμο randomness και

chromo\_liver-=randomness #Αφαιρουμε απο την αλληλουχια 2 τον τυχαιο αριθμο randomness

check(player)

**if**(player==**1**): #Αλλαγη παιχτων

player=**2** #Αν επαιζε ο παιχτης 1 τωρα εχει σειρα ο παιχτης 2

**elif**(player==**2**): #Αλλαγη παιχτων

player=**1** #Αν επαιζε ο παιχτης 2 τωρα εχει σειρα ο παιχτης 1

**else**: #Αν δεν μπει στην if

**break**; #Βγαινει απο την while

Παράδειγμα εκτέλεσης:

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

*Επεξήγηση του προγράμματος:*

Το χρωμόσωμα liver έχει αρχικώς 39314 νουκλεοτίδια και το χρωμόσωμα brain έχει αρχικαώς 49943 νουκλεοτίδια.

Σε πρώτο στάδιο, θα παίξει ο παίκτης 1 και θα επιλέγει αν θέλει να διαγράψει από το πρώτο χρωμόσωμα ή από το δεύτερο ή και από τα 2 χρωμοσώματα.

Στη συνέχεια, θα παίξει ο παίκτης 2 και θα επιλέγει αντίστοιχα από πού θέλει να διαγράψει έναν τυχαίο αριθμό νουκλεοτιδιών.  
Το παιχνίδι θα τελειώσει όταν ο αριθμός των νουκλεοτιδιών σε ένα από τα 2 χρωμοσώματα θα γίνει ίσος με το 0.

**4. ΘΕΜΑ 4ο**

*4.1 Ερώτημα 1ο*

α) Η μέθοδος με την οποία εχει προσδιορισθέι η δομή του συμπλόκου είναι η : **X-RAY DIFFRACTION**

β) Η διακριτική ικανότητα της δομής είναι 1.77

γ) To DOI της δημοσίευσης είναι: 10.1016/j.cell.2021.02.033

*4.2 Ερώτημα 2ο*

α) Περιλαμβάνει 3 διακριτές πρωτεϊνικές αλυσλίδες

β) COVOX-269 Fab heavy chain : το πλήθος των αμινοξέων είναι 222 COVOX-269 fab light chain : το πλήθος των αμινοξέων είναι 215 Spike glycoprotein : το πλήθος των αμινοξέων είναι 205

γ) Περιέχει έναν ολιγοσακχαρίτη τον: 2-acetamido-deoxy-beta-D-glucopyranose-(1-4)-[alpha-L-fucopyranose-(1-6)]2-acetamido-2-deoxy-beta-D-glucopyranose

δ) Ανήκει στην αλυσίδα CA

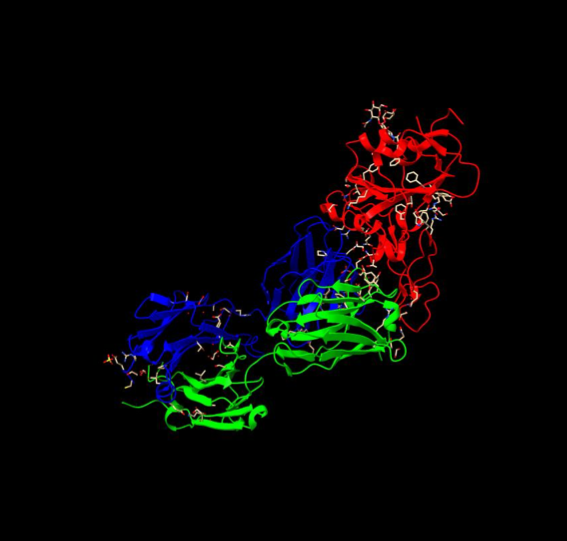
*4.3 Ερώτημα 3ο*

α)

Εικόνα που περιέχει κείμενο

Περιγραφή που δημιουργήθηκε αυτόματα

β)



*4.4 Ερώτημα 4ο*

α)

Εικόνα που περιέχει κοιλεντερωτό, υδρόζωο

Περιγραφή που δημιουργήθηκε αυτόματα

*4.5 Ερώτημα 5ο*

β)



γ)

