ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΑΠΑΛΛΑΚΤΙΚΗ ΕΡΓΑΣΙΑ ΒΙΟΠΛΗΡΟΦΟΡΙΚΗΣ 2019

ΚΑΘΗΓΗΤΗΣ : Α. ΠΙΚΡΑΚΗΣ

ΣΤΟΙΧΕΙΑ ΦΟΙΤΗΤΩΝ

ΒΕΡΓΙΑΝΝΗΣ ΝΙΚΟΛΑΟΣ - Π16170

ΜΗΝΑΔΑΚΗΣ ΓΕΩΡΓΙΟΣ - Π13084

ΓΡΑΜΜΕΝΟΣ ΓΕΡΑΣΙΜΟΣ - Π16025

ΕΚΦΩΝΗΣΗ ΕΡΓΑΣΙΑΣ:

- Η εργασία είναι απαλλάκτική και εκπονείται σε ομάδες των 1-3 φοιτητών. Όσοι φοιτητές δεν λάβουν προβιβάσιμο βαθμό, εξετάζονται και βαθμολογούνται γραπτώς τον Σεπτέμβριο, ομοίως και όσοι δεν υποβάλουν εργασία.

- Η εργασία παραδίδεται με email στο pikrakis@unipi.gr, έως και την Κυριακή 30/6, ώρα 23:59. Μπορείτε να στείλετε dropbox link, ΜΟΝΟ εάν τα επισυναπτόμενα υπερβαίνουν σε όγκο τη δυνατότητα αποστολής με email.

- Υλοποιήστε 7 (επτά) εκ των παρακάτω ασκήσεων του βιβλίου:

6.12, 6.13, 6.14, 6.15, 6.16 (δύο)

6.22, 6.23, 6.27, 6.37, 6.51 (τρεις)

11.4, 11.6 (δύο)

- Οι ασκήσεις είναι ισοδύναμες. Αποδεκτές γλώσσες υλοποίησης είναι μόνο οι Python, Octave/Matlab, C/C++. Κάθε άσκηση συνοδεύεται από τεκμηρίωση της λύσης. Ορισμένες εκ των ασκήσεων συσχετίζονται με επιπρόσθετα δεδομένα, για τις περιγραφές των οποίων ανατρέξτε στην ενότητα εγγράφων του μαθήματος.

- Ακολουθήστε τις οδηγίες που έχω δώσει για επιτυχή εκπόνηση. Θα γίνει αυστηρός έλεγχος για να διαπιστωθούν πιθανές περιπτώσεις αντιγραφής.

Όλες οι εργασίες εχουν υλοποιηθεί σε ***Python*** *με το* ***PyCharm(IDE).***

Χρήσιμη θεωρεία για την υλοποίηση των εργασίων

Στοίχιση ακολουθιών:

Η στοίχιση ακολουθιών (sequence alignment), είναι μια διαδικασία κατά την οποία δύο ακολουθίες ή αλλιώς συμβολοσειρές τοποθετούνται η μία κάτω από την άλλη, με τέτοιον τρόπο που τα κοινά τους σύμβολα να είναι τοποθετημένοι στην ίδια θέση. Σκοπός είναι να βρεθεί η "βέλτιστη στοίχιση", δηλαδή η στοίχιση στην οποία οι δύο ακολουθίες ταιριάζουν περισσότερο μεταξύ τους. Για να γίνει αυτό, χρησιμοποιείται κάποιο μοτίβο βαθμολογίας, και η βέλτιστη στοίχιση είναι αυτή που συγκεντρώνει την υψηλότερη βαθμολογία. Για παράδειγμα, κάθε ζεύγος συμβόλων που ταιριάζει, βαθμολογείται με κάποιο θετικό σκορ (π.χ. +1), ενώ κάθε ζεύγος που διαφέρει βαθμολογείται με κάποιο αρνητικό σκορ (π.χ. -1). Αν απαιτείται να εισαχθεί κάποιο κενό έτσι ώστε οι ακολουθίες να ταιριάζουν σε άλλα σημεία, το κάθε κενό βαθμολογείται πάλι με κάποιο αρνητικό σκορ (π.χ. -1). Το άθροισμα όλων των επιμέρους 13 σκορ, μας δίνει την συνολική βαθμολογία της ταξινόμησης. Ελέγχοντας όλες τις δυνατές ταξινομήσεις και επιλέγοντας αυτήν με την μεγαλύτερη βαθμολογία, παίρνουμε την βέλτιστη στοίχιση.

Η στοίχιση ακολουθιών χρησιμοποιείται στο πεδίο της βιοπληροφορικής για την εύρεση ομοιοτήτων μεταξύ βιολογικών ακολουθιών (DNA, RNA, πρωτεΐνες...), μια εργασία πολύ συχνή και με ιδιαίτερη σημασία. Η σημασία της πηγάζει από το γεγονός ότι οι ομοιότητες αυτές υποδηλώνουν συνήθως κάποια βιολογική συσχέτιση, πράγμα που οδηγεί σε καλύτερη κατανόηση των διαφόρων βιολογικών μηχανισμών.

Υπάρχουν δύο είδη στοίχισης, η **ολική στοίχιση** (global alignment) και η **τοπική στοίχιση** (local alignment), οι οποίες χρησιμοποιούνται ανάλογα με τον τύπο του προβλήματος που αντιμετωπίζεται.

Ολική στοίχιση ακολουθιών:

Κατά την ολική στοίχιση ακολουθιών (global sequence alignment) γίνεται προσπάθεια να στοιχιστεί κάθε σύμβολο κάθε ακολουθίας. Οι δύο ακολουθίες στοιχίζονται σε ολόκληρο το μήκος τους με τον καλύτερο δυνατό τρόπο. Κάθε σύμβολο κάθε ακολουθίας, λοιπόν, αντιστοιχίζεται σε ένα σύμβολο της άλλης ή σε ένα κενό. Ένας γνωστός αλγόριθμος για ολική στοίχιση ακολουθιών είναι ο αλγόριθμος **Needleman-Wunsch**.

Τοπική στοίχιση ακολουθιών:

Κατά την τοπική στοίχιση ακολουθιών (local sequence alignment) γίνεται η καλύτερη δυνατή στοίχιση μεταξύ τμημάτων των δύο ακολουθιών. Δηλαδή επιτρέπεται κάποια κομμάτια που "χαλούν" τη στοίχιση να μείνουν εκτός. Ένα παράδειγμα που χρησιμοποιείται συχνά είναι ότι μπορούμε να χρησιμοποιήσουμε την τοπική στοίχιση ακολουθιών προκειμένου να βρούμε τις προτάσεις δύο κειμένων, οι οποίες παρουσιάζουν την περισσότερη ομοιότητα. Ένας γνωστός αλγόριθμος για τοπική στοίχιση ακολουθιών είναι ο αλγόριθμος **Smith-Waterman.**

Αλγόριθμος Needleman – Wunsch:

Ο αλγόριθμος Needleman-Wunsch είναι ένας αλγόριθμος που χρησιμοποιείται για ολική στοίχιση ακολουθιών. Συγκεκριμένα χρησιμοποιείται για να υπολογιστεί η βέλτιστη ολική στοίχιση μεταξύ δύο ακολουθιών - συμβολοσειρών. Κατά την ολική στοίχιση ακολουθιών (global sequence alignment) γίνεται προσπάθεια να στοιχιστεί κάθε σύμβολο κάθε ακολουθίας. Οι δύο ακολουθίες στοιχίζονται σε ολόκληρο το μήκος τους με τον καλύτερο δυνατό τρόπο. Κάθε σύμβολο κάθε ακολουθίας, λοιπόν, αντιστοιχίζεται σε ένα σύμβολο της άλλης ή σε ένα κενό. Η στοίχιση γίνεται με την χρήση ενος δισδιαστατου πίνακα όπου κάθε κελί αντιστοιχεί στο ταίρασμα καθε γράμματος απο κάθε ακολουθία. Η στοίχιση ξεκινά απο πάνω αριστερά και ακολουθεί κυρίως ένα διαγώνιο μονοπάτι προς τα κάτω και δεξία κρατώντας ενα σκορ(1 για ταίριασμα και -1 για την ασυμφωνία και το κενό).

Ο πίνακας D κατατάσσεται και χτίζεται αναδρομικά:

$$D(i, j) = \max \left\{ \; D(I-1, j-1) + s(xi, yj) \;,\; D(i-1, j) + g \;,\; D(i, j-1) + g \; \right\}$$

Αλγόριθμος Smith-Waterman:

Ο αλγόριθμος Smith-Waterman είναι ένας αλγόριθμος που χρησιμοποιείται για τοπική στοίχιση ακολουθιών. Για να το επιτύχει αυτό συγκρίνει τμήματα όλων των πιθανών μηκών και βελτιστοποιεί την βαθμολογία στοίχισης, που αποτελεί μέτρο για την ομοιότητα των ακολουθιών. Είναι παραλλαγή του αλγορίθμου Needleman-Wunch με την διαφορά ότι δεν χρησιμοποιεί αρνητικές βαθμολογίες 16 (όταν ένα ζεύγος έχει αρνητική βαθμολογία, τότε αυτή τίθεται ίση με μηδέν). Χρησιμοποιείται για να αναγνωρίσει τις ομοιότητες μεταξύ δύο ακολουθιών. Για να το επιτύχει αυτό δρα σε 2 στάδια. Αρχικά υπολογίζει έναν πίνακα βαθμολογίας, που περιέχει όλες τις πιθανές συγκρίσεις. Έπειτα, ξεκινώντας από το στοιχείο του πίνακα με την υψηλότερη βαθμολογία κάνει οπισθοδρόμηση για να βρει την ακριβής στοίχιση των δύο ακολουθιών που συγκρίνει.

$$\mathbf{H\,i\,,\,j = Max} \left\{ \; \mathbf{Max(\,H\,i-1,\,j-1+S\,i\,,\,j,\,0)} \;\; \mathbf{Max0<k<j(H\,i\,,\,j-k-(Gs+kGe))} \right.$$

$$\left. \mathbf{Max0<k<i(H\,i-k\,,\,j-(Gs+kGe))} \; \right\}$$

Άσκηση 6.12

Δύο παίκτες παίζουν το παρακάτω παιχνίδι με δύο «χρωμοσώματα» που έχουν μήκος n και m νουκλεοτίδια αντίστοιχα. Σε κάθε γύρο του παιχνιδιού, ένας παίκτης μπορεί να καταστρέψει ένα απο τα χρωμοσώματα και να διαχωρίσει το άλλο σε δύο μή κενά τμήματα. Για παράδειγμα, ο πρώτος παίκτης μπορεί να καταστρέψει ένα χρωμόσωμα μήκος n και να διαχωρίσει ένα άλλο χρωμόσωμα σε δύο χρωμοσώματα με μήκη m/3 και m-m/3. Ο παίκτης που διαγράφει το τελευταίο νουκλεοτίδιο κερδίζει. Ποιος θα κερδίσει; Περιγράψτε τη νικηφόρα στρατηγική για όλες τις τιμές των n και m.

Λύση

Το πρόγραμμα είναι μια παρτίδα παιχνιδιού ανάμεσα στον χρήστο και στο cpu. Το πρόγραμμα λειτουργεί ανάλογα με τις επιλογές του χρήστη. Σαν πρώτη επιλογή διαλέγει αν θα παίξει πρώτος εκείνος. Ο χρήστης διαγράφει ένα χρωμόσωμα και διαχωρίζει το άλλο σε άλλα δυο με μήκη m/3 και m-m/3. Έπειτα παίζει ο υπολογιστής κάνωντας το ίδιο.

Νουκλεοτίδιο Liver = https://www.ncbi.nlm.nih.go v/nuccore/NC_000014.9?rep ort=fasta&from=50905217&t o=50944530&strand=true

Μήκος Liver : `39314`

Νουκλεοτίδιο Muscle = https://www.ncbi.nlm.nih.go v/nuccore/NC_000011.10?re port=fasta&from=64746389& to=64760715&strand=true

Μήκος Muscle: `14327`

Η ολοκλήρωση κάθε γύρου του παιχνιδιού εξαρτάται αποκλειστικά από τον τρόπο που επιλέγουμε να διαχωριστεί το χρωμόσωμα που δε διαγράφεται (δηλαδή από τον παρονομαστή) και τις κινήσεις που θα επιλέξει να κάνει ο κάθε παίκτης. Μια νικηφόρα στρατηγική για τον κάθε παίκτη είναι να υπάρχει ένα και μόνο ένα χρωμόσωμα στη σειρά του (αυτό που θα διαγράψει για να κερδίσει) καθώς το άλλο απαλείφεται κατά κάποιο διαχωρισμό.

Κώδικας:

```
        # an numa>nub tote paizei o upologisths kai diagrafei to numa
        elif (int(numA) >= int(numB)):
            numToDelete = int(numA)
            numA = int(numB) / 3
            numB = int(numB) - int(numB) / 3
            print("Computer deleted chromosome A.")
            print("Chromosome A turned to be ", int(numA), " and B ", int(numB), ".")

        # an numb?numa tote paizei o upologisths kai diagrafei to numb
        elif (int(numA) <= int(numB)):
             numToDelete = int(numB)
             numB = int(numA) - int(numA) / 3
             numA = int(numA) / 3
             print("Computer deleted chromosome B.")
             print("Chromosome A turned to be ", int(numA), " and B ", int(numB), ".")

        # an einai kai ta 2 xrwmoswmata 0 tote o upologisths kerdizei
        if  (int(numA) == 0 and int(numB) == 0):
                print("you lost")
                break

    # an o xrhsths pathsei 'n' tote ksekinaei o upologisths
    if(input3 == "n"):
         print("Computer is playing!(", int(numA), " or ", int(numB), ").")
         if int(numA) <= int(numB):   #an numa<=numb tote o upologisths diagrafei to numb
             numToDelete = int(numB)
             numB = int(numA) - int(numA) / 3
             numA = int(numA) / 3
             print("Computer deleted chromosome B.")
             print("Chromosome A turned to be ", int(numA), " and B ", int(numB), ".")

         elif(int(numA) >= int(numB)):  #an numa>=numb tote o upologisths diagrafei to numa
             numToDelete = int(numA)
             numB = int(numB) - int(numB) / 3
             numA = int(numB) / 3
             print("Computer deleted chromosome A.")
             print("Chromosome A turned to be ", int(numA), " and B ", int(numB), ".")
```

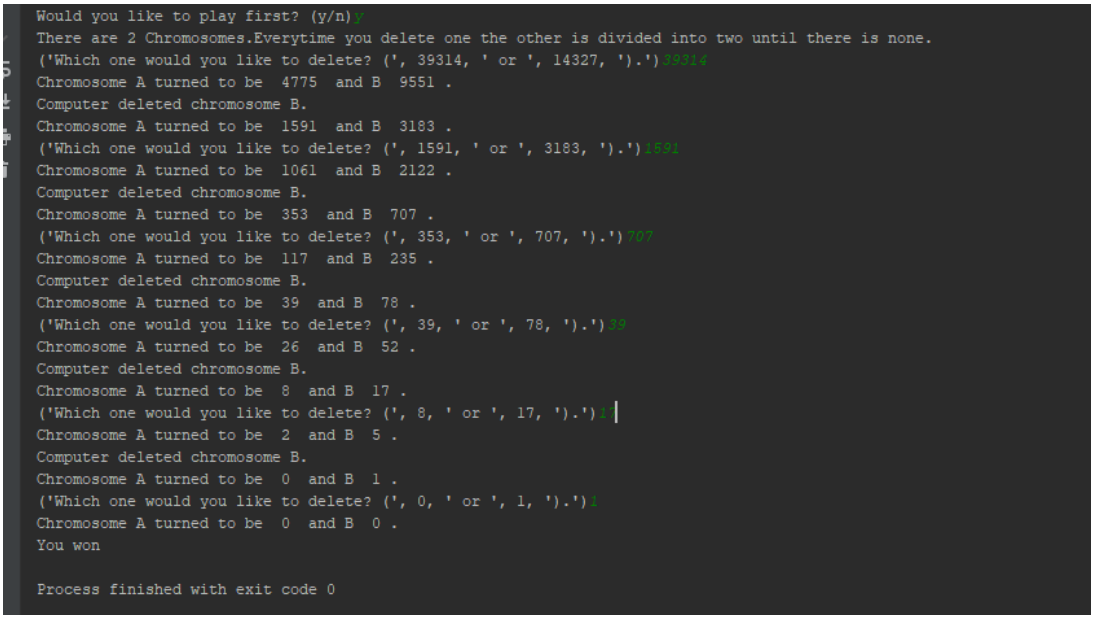
```
        # an kai ta 2 xrwmoswmata einai 0 tote kerdizei o upologisths
        if (int(numA) == 0 and int(numB) == 0):
            print("You lost")
            break

        # rwtaei ton xrhsth poio apo ta 2 xrwmoswmata thelei na diagrapsei
        input1=input(("Which one would you like to delete? (", int(numA), " or ", int(numB), ")."))
        numToDelete=int(input1)
        # an epileksei to nua tote to numa diaireitai me to 3,kai sthn thesh tou numb mpainei to nua-numa/3
        if(numToDelete == int(numA)):
            numA = int(numB) / 3
            numB = int(numB) - int(numB) / 3
            print("Chromosome A turned to be ", int(numA), " and B ", int(numB), ".")

        # an epileksei to nub tote to numb diairetai me to 3 ,kai sthn thesh tou numb mpainei to nub-nub/3
        elif(numToDelete == int(numB)):
            numB = int(numA) - int(numA) / 3
            numA = int(numA) / 3
            print("Chromosome A turned to be ", int(numA), " and B ", int(numB), ".")

        if(int(numA) == 0 and int(numB) == 0):  # an kai ta 2 xrwmoswmata exoun mhdenistei tote kerdizei o xrhsths
            print("you won")
            break
```

Παράδειγμα για νουκλεοτίδιο n = Liver και νουκλεοτίδιο m = Muscle :



Άσκηση 6.15

Δύο παίκτες παίζουν το παρακάτω παιχνίδι με μια νουκλεοτιδική αλληλουχία που έχει μήκος n. Σε κάθε γύρο του παιχνιδιού, ένας παίκτης μπορεί να αφαιρέσει είτε ένα είτε δύο νουκλεοτίδια από την αλληλουχία. Ο παίκτης που αφαιρεί το τελευταίο γράμμα κερδίζει. Ποιος θα κερδίσει; Περιγράψτε τη νικηφόρα στρατηγική για όλες τις τιμές του n.

Λύση

Η αλληλουχία νουκλεοτιδίων για την παρούσα άσκηση πάρθηκε από την

α-λακταλβουμίνη = https://www.ncbi.nlm.nih.go v/nuccore/AC_000162.1?repo rt=fasta&from=31347861&to =31349882.

Μήκος **α-λακταλβουμίνη**= 2022.

Στόχος του κάθε παίκτη είναι να αναγκάζει τον αντίπαλό του να αφαιρεί από μονό αριθμό ώστε εκείνος να αφαιρεί από ζυγό, ώστε να αφαιρέσει τα δυο τελευταία γράμματα, που θα τον οδηγήσει στη νίκη.

Κώδικας:

```
numLetters = 30
numToTake = 0
numToTakePC = 1
input3 = input(print("Would you like to play first?(y/n)"))


while numLetters > 0:
    if (input3 == "y"):  # an o xrhsths pathsei y tote tha paiksei prwtos
        print("\nThere are ", numLetters, " letters.")
        numToTake = input(print("How many letters to take? (1 or 2)"))
        numToTake = int(numToTake)

        if (numToTake >= 2):  # an o xrhsths pathsei to 2 tote sto numtotake tha apothkeutei to 2
            numToTake = 2

        elif numToTake <= 1:  # antistoixa an pathsei 1
            numToTake = 1

        numLetters = numLetters - numToTake  # analoga me to ti exei dwsei ws input o xrhsths,tha afairethei apo to numbletters
```

```
        if numLetters <= 0:  # an den uparxoun alla numletters tote o xrhsths kerdizei
            print("You won!")

        elif ((numLetters - numToTakePC) <= 0 or numLetters <= 0):  # an o upologisths kerdizei afairwntas 2 tote afairei 2
            numToTakePC = 2
            print("Computer takes ", numToTakePC, " letters.")
            print("You lost")

        else:  # alliws afairei 1
            numToTakePC = 1
            print("Computer takes ", numToTakePC, " letters.")
            if (numLetters <=0):
                print("You lost")

        numLetters = numLetters - numToTakePC
```

```
    elif(input3 == "n"):  # an paizei deuteros o xrhsths

        print("There are ", numLetters, " letters.")

        numLetters = numLetters - numToTakePC
        print("\nComputer takes ", numToTakePC, " letters.")

        print("There are ", numLetters, " letters.")
        numToTake = input(print("\nHow many letters to take? (1 or 2)"))
        numToTake = int(numToTake)

        if (numToTake == 2):  # an o xrhsths pathsei to 2 tote sto numtotake tha apothkeutei to 2
            numToTake = 2
            numToTakePC = 1

            numLetters = numLetters - numToTake
```

```
        elif(numToTake == 1):  # antistoixa an pathsei 1
            numToTake = 1
            numToTakePC = 2

            numLetters = numLetters - numToTake  # analoga me to ti exei dwsei ws input o xrhsths,tha afairethei apo to numbletters



        if numLetters == 0:  # an den uparxoun alla numletters tote o xrhsths kerdizei
            print("You won!")

        elif ((numLetters - numToTakePC) == 0 or numLetters < 0):  # an o upologisths kerdizei afairwntas 2 tote afairei 2

            print("You lost")

    else:
        print("Parakalw eisagete mono y/n")
```

Εκτέλεση παραδείγματος ξεκινώντας πρώτος σαν χρήστης:

```
Would you like to play first?(y/n)
Noney

There are  30  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  28  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  26  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  24  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  22  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  20  letters.
How many letters to take? (1 or 2)
```

```
There are  18  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  16  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  14  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  12  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  10  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.
```

```
There are  8  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  6  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  4  letters.
How many letters to take? (1 or 2)
None1
Computer takes  1  letters.

There are  2  letters.
How many letters to take? (1 or 2)
None2
You won!
```

Εκτέλεση παραδείγματος όταν ξεκινάει το PC:

```
C:\Users\Nikos\PycharmProjects\Unnamed\venv\Scripts\python.exe C:/Users/Nikos/Downloads/6.15.py
Would you like to play first?(y/n)
None n
There are  30  letters.

Computer takes  1  letters.
There are  29  letters.

How many letters to take? (1 or 2)
None 2
There are  27  letters.

Computer takes  1  letters.
There are  26  letters.

How many letters to take? (1 or 2)
None 1
There are  25  letters.

Computer takes  2  letters.
There are  23  letters.

How many letters to take? (1 or 2)
None 2
There are  21  letters.

Computer takes  1  letters.
There are  20  letters.

How many letters to take? (1 or 2)
None 1
There are  19  letters.

Computer takes  2  letters.
There are  17  letters.

How many letters to take? (1 or 2)
None 2
There are  15  letters.
```

```
There are  15  letters.

Computer takes  1  letters.
There are  14  letters.

How many letters to take? (1 or 2)
None 1
There are  13  letters.

Computer takes  2  letters.
There are  11  letters.

How many letters to take? (1 or 2)
None 2
There are  9  letters.

Computer takes  1  letters.
There are  8  letters.

How many letters to take? (1 or 2)
None 1
There are  7  letters.

Computer takes  2  letters.
There are  5  letters.

How many letters to take? (1 or 2)
None 2
There are  3  letters.

Computer takes  1  letters.
There are  2  letters.

How many letters to take? (1 or 2)
None 2
You won!
```

Άσκηση 6.22:

Ορίζουμε οτι η στοίχιση επικάλυψης μεταξύ δύο αλληλουχιών $v = v_1 \ldots v_n$ *και* $w = w_1 \ldots w_m$ *είναι η στοίχιση ανάμεσα σε ένα πρόθεμα της v και ένα επίθεμα της w. Για παράδειγμα, αν v = TATATA και w = AAATTT,τότε μια (όχι απαραιτήτως βέλτιστη) στοίχιση επικάλυψης μεταξύ των v και w είναι η*

ATA

AAA

Η βέλτιστη στοίχιση επικάλυψης είναι η στοίχιση που μεγιστοποιεί τη βαθμολογία της καθολικής στοίχισης μεταξύ των v1,...,vn της v και όλα τα επιθέματα w1,...,wj της w.

Πρόβλημα:

Διατυπώστε έναν αλγόριθμο που υπολογίζει τη βέλτιστη στοίχιση επικάλυψης και εκτελείται σε χρόνο 0(n.m)

Λύση

Για την λύση αυτής της άσκησης χρησιμοποίησαμε τον αλγόριθμο **Needleman – Wunsch** με μια μικρη παραλλαγή.

Φτιάξαμε έναν πίνακα array ο οποίος παίρνει 2 ακολουθίες μήκους 6 π.χ.

(v= TATATA και w = AAATTT)

και μετατρέπει τον 2ο στο αντιστροφο του (wrvs=TTTAAA) στοιχίζοντας το v οριζόντια και το νεο w κάθετα. Ο πίνακας γίνεται ως εξής:

0	T	A	T	A	T	A
T	1	0	-1	-2	-3	-4
T	0	0	-1	0	-1	-2
T	-1	1	1	0	-1	-2
A	-2	0	0	2	1	0
A	-3	-1	1	1	1	0
A	-4	-2	0	2	2	2

Για τα στοιχεία όπου i= j κρατάμε το σκορ και το βάζουμε σε μια καινούργια λίστα elements και απο εκεί κρατάμε το μέγιστο(max) στοιχείο της λίστας όπου είναι και η βέλτιστη στοίχιση επικάλυψης. Στην περίπτωση αυτή είναι το πρόθεμα v= TATA και το επίθεμα w=TTTA αλλά και ολόκληρη η v με την w. Προτιμάμε να κρατάμε το σκορ με τα λιγότερα στοιχεία.

Κώδικας:

```python
import numpy as np

elements=[]
match_score = 1
mismatch_score = -1
gap_penalty = -1


sequence_1 = "TATATA"
sequence_2 = "AAATTT"

sequence_2rvs = sequence_2[::-1]


seq1_list=list(sequence_1)
seq2_list=list(sequence_2rvs)


len_1 = len(sequence_1)
len_2 = len(sequence_2rvs)

array = np.zeros(shape=(len_1 + 1, len_2 + 1))


def p(i, j):
    if sequence_1[i] == sequence_2rvs[j]:
        return match_score
    else:
        return mismatch_score
```

```python
for i in range(0, len(sequence_1) + 1):
    for j in range(0, len(sequence_2rvs) + 1):
        if i == 0:
            array[0, j] = gap_penalty * j
        elif j == 0:
            array[i, 0] = gap_penalty * i
        else:
            array[i, j] = max(array[i - 1, j - 1] + p(i - 1, j - 1), array[i - 1, j] + gap_penalty,
                              array[i, j - 1] + gap_penalty)

print(array)

for i in range(0,len_1):
    elements.insert(i-1,array[i][i])
```

```
    if array[1][1]==max(elements):
        print("H beltisth stoixish epikalupshs epitugxanetai me to prothema "+seq1_list[0] +
              " kai to epithema  : "+seq2_list[0]+"")

    elif array[2][2]==max(elements):
        print("H beltisth stoixish epikalupshs epitugxanetai me to prothema "+seq1_list[0]+seq1_list[1]
              + " kai to epithema : "+seq2_list[0]+seq2_list[1]+"")


    elif array[3][3]==max(elements):
        print("H beltisth stoixish epikalupshs epitugxanetai me to prothema "+seq1_list[0]+seq1_list[1]+seq1_list[2]
              + " kai to epithema : "+seq2_list[0]+seq2_list[1]+seq2_list[2]+"")


    elif array[4][4]==max(elements):
        print("H beltisth stoixish epikalupshs epitugxanetai me to prothema "+seq1_list[0]+seq1_list[1]+seq1_list[2]+seq1_list[3]
              +" kai to epithema : "+seq2_list[0]+seq2_list[1]+seq2_list[2]+seq2_list[3]+"")


    elif array[5][5]==max(elements):
        print("H beltisth stoixish epikalupshs epitugxanetai me to prothema "+seq1_list[0]+seq1_list[1]+seq1_list[2]+seq1_list[3]
              +seq1_list[4]
              + " kai to epithema : "+seq2_list[0]+seq2_list[1]+seq2_list[2]+seq2_list[3]+seq2_list[4]+"")


    elif array[6][6]==max(elements):
        print("H beltisth stoixish epikalupshs epitugxanetai me to prothema "+seq1_list[0]+seq1_list[1]+seq1_list[2]+seq1_list[3]
              +seq1_list[4]+seq1_list[5]
              +" kai to epithema : "+seq2_list[0]+seq2_list[1]+seq2_list[2]+seq2_list[3]+seq2_list[4]+seq2_list[5]+"")
```

Εκτέλεση παραδείγματος βιβλίου:

```
C:\Users\Nikos\PycharmProjects\untitled\venv\Scripts\python.exe "C:/Users/Nikos/Desktop/ΠΑΠΕΙ/6
[[ 0. -1. -2. -3. -4. -5. -6.]
 [-1.  1.  0. -1. -2. -3. -4.]
 [-2.  0.  0. -1.  0. -1. -2.]
 [-3. -1.  1.  1.  0. -1. -2.]
 [-4. -2.  0.  0.  2.  1.  0.]
 [-5. -3. -1.  1.  1.  1.  0.]
 [-6. -4. -2.  0.  2.  2.  2.]]
H beltisth stoixish epikalupshs epitugxanetai me to prothema TATA kai to epithema : TTTA

Process finished with exit code 0
```

Εκτέλεση παραδείγματος με 2 τυχαιες ακολουθίες v= TGCTGC και w= GGCCGG:

```
C:\Users\Nikos\PycharmProjects\untitled\venv\Scripts\python.exe "C:/Users/Nikos/Desktop/
[[ 0. -1. -2. -3. -4. -5. -6.]
 [-1. -1. -2. -3. -4. -5. -6.]
 [-2.  0.  0. -1. -2. -3. -4.]
 [-3. -1. -1.  1.  0. -1. -2.]
 [-4. -2. -2.  0.  0. -1. -2.]
 [-5. -3. -1. -1. -1.  1.  0.]
 [-6. -4. -2.  0.  0.  0.  0.]]
H beltisth stoixish epikalupshs epitugxanetai me to prothema TGC kai to epithema : GGC

Process finished with exit code 0
```

Άσκηση 6.23

Διατυπώστε έναν αλγόριθμο που υπολογίζει τη βέλτιστη στοίχιση προσαρμογής. Εξηγήστε πώς συμπληρώνεται η πρώτη γραμμή και η πρώτη στήλη του πίνακα δυναμικού προγραμματισμού και γράψτε μια σχέση επανάληψης για τη συμπλήρωση του υπόλοιπου πίνακα. Παρουσιάστε μια μέθοδο που βρίσκει την καλύτερη στοίχισει αφού συμπληρωθεί ο πίνακας. Ο αλγόριθμος θα πρέπει να εκτελείται σε χρόνο 0(nm)

Λύση:

Δημιουργήσαμε εναν αλγόριθμο οπου αρχικοποιούμε 2 πίνακες array και backtrack και στον πίνακα array θα γινει η καθολική στοιχίση των 2 αλληλουχιών με τα σκορ (1 για ταίρασμα, -1 για ασυμφωνία ή προσθαφαίρεση) και στον πινακα backtrack θα γίνει αναδρομη προς τα πίσω οταν τελειώνει η μικρότερη αλληλουχία w. Χρησιμοποίησαμε μια ανωνυμη lambda function για την εισχώρηση κενού κρατώντας μονο το ταιριάσματα με σκορ 2 στον πίνακα backtrack. Έπειτα επιστρέφουμε την καλύτερη βαθμολογία (max_score) και τις αλληλουχίες v και w στοιχισμένες βαση της στοίχισης προσαρμογής.

Array:

-	T	A	G	A	T	A
G	-1	-1	1	0	-1	-1
T	1	0	0	0	1	0
A	0	2	1	1	0	2
G	-1	1	3	2	1	1
G	-1	0	2	2	1	0
C	-1	-1	1	1	1	0
T	1	0	0	0	2	1
T	0	1	0	-1	1	1
A	0	2	1	0	0	2
A	-1	1	1	2	1	1
G	-1	0	2	1	1	0
G	-1	-1	1	1	0	0
T	1	0	0	0	2	1
T	1	0	-1	-1	1	1
A	0	2	1	0	0	2

Backtrack:

-	T	A	G	A	T	A
G	-	-	2	-	-	-
T	2	-	-	2	2	-
A	-	↖2	-	2	-	2
G	-	-	↖2	-	-	-
G	-	-	-	↖2	-	-
C	-	-	-	↑-	2	-
T	2	-	-	-	↖2	-
T	2	-	-	-	↑-	2
A	-	2	-	-	-	↖2
A	-	-	2	2	-	-
G	-	-	2	-	2	-
G	-	-	-	2	-	2
T	2	-	-	-	2	-
T	2	-	-	-	-	2
A	-	2	-	-	-	2

Κώδικας:

```
seq1 = "GTAGGCTTAAGGTTA"
seq2 = "TAGATA"

def fitting_alignment(v,w):

    # Arxikopoihsh pinakwn.
    array = [[0 for j in xrange(len(w)+1)] for i in xrange(len(v)+1)]
    backtrack = [[0 for j in xrange(len(w)+1)] for i in xrange(len(v)+1)]

    # Symplhrwsh pinakwn array kai backtrack.
    for i in xrange(1, len(v)+1):
        for j in xrange(1, len(w)+1):
            scores = [array[i-1][j] - 1, array[i][j-1] - 1, array[i-1][j-1] + [-1, 1][v[i-1] == w[j-1]]]
            array[i][j] = max(scores)
            backtrack[i][j] = scores.index(array[i][j])

    # Thesh me to megalytero keli poy antistoixei sto telos tis mikroterhs allhlouxias w
    j = len(w)
    i = max(enumerate([array[row][j] for row in xrange(len(w), len(v))]),key=lambda x: x[1])[0] + len(w)
    max_score = str(array[i][j])

    # Arxikopoihsh eythigrammismenwn allhlouxiwn sthn thesh me to megalitero score.
    v_aligned, w_aligned = v[:i], w[:j]

    # Lambda function gia eisxwrisi kenou '-'.
    insert_gap = lambda word, i: word[:i] + '-' + word[i:]
```

```
28
29      # Anatrexoume ton pinaka gia na ksekinisei h stoixisi prosarmoghs.
30      # opou 0 = - sthn w (gap)
31      # opou 1 = - sthn v (gap)
32      while i*j != 0:
33          if backtrack[i][j] == 0:
34              i -= 1
35              w_aligned = insert_gap(w_aligned, j)
36          elif backtrack[i][j] == 1:
37              j -= 1
38              v_aligned = insert_gap(v_aligned, i)
39          elif backtrack[i][j] == 2:
40              i -= 1
41              j -= 1
42
43
44          # Stamatame sto teleutaio shmeiou tou backtrack.
45          v_aligned = v_aligned[i:]
46
47
48
49      return max_score, v_aligned, w_aligned
50
51
52  print '\n'.join(fitting_alignment(seq1,seq2))
53
```

Εκτέλεση παραδείγματος Βιβλίου:

```
C:\Users\Nikos\PycharmProjects\untitled1\venv\Scripts\python.exe "
2
TAGGCTTA
TAGA-T-A

Process finished with exit code 0
```

Εκτέλεση παραδείγματος με 2 τυχαιες ακολουθίες :

v = TATCCTGCAAGTTACATTCA και w = TACGCA

```
C:\Users\Nikos\PycharmProjects\untitled1\venv\Scripts\python.e
3
TATCCTGCA
TA-C--GCA

Process finished with exit code 0
```

Ασκηση 6.37

Επινοήστε έναν αποδοτικό αλγόριθμο για το πρόβλημα της Χιμαιρικής Στοίχισης.

Ένας ιός μολύνει ένα βακτήριο και τροποποιεί μια διεργασία αναδιπλασιασμού στο βακτήριο προσθέτοντας

Σε καθε A, ενα πολυA με μήκος απο 1 έως 5

Σε καθε C, ενα πολυC με μήκος 1 έως 10

Σε κάθε G, ένα πολυG με τυχαίο μήκος ≥ 1

Σε κάθε T, ένα πολυT με τυχαίο μήκος ≥ 1.

Δεν επιτρέπονται κενά ή άλλες προσθήκες στο DNA που έχει τροποποιηθεί απο τον ιό. Για παράδειγμα, η αλληλουχία AAATAAAGGGGCCCCCTTTTTTTCC αποτελεί μολυσμένη έκδοση της ATAGCTC

Λύση:

Έχουμε την μολυσμένη εκδοση v= *AAATAAAGGGGCCCCCTTTTTTTCC* την οποία την προσθέτουμε σε μία λιστα (list1) και ψάχνουμε να βρούμε την κανονική της μορφή η οποία αποθηκεύεται σε μια δεύτερη λίστα. Χρησιμοποιήσαμε while και if loops, τα οποια ελέγχουν απο την μολυσμένη έκδοση το πρώτο και το δεύτερο γράμμα και αν είναι το ίδιο διαγράφει έως ότου βρει διαφορετικό γράμμα όπου και το προσθέτει στην δεύτερη λίστα(list2). Για τους αναδιπλασιασμούς χρησιμοποιήσαμε counters ώστε να καλύπτονται οι περιορισμοί(π.χ.πολυA μήκος 1~5 κλπ).

Κώδικας:

```
input="AAATAAAGGGGCCCCCTTTTTTTCC"
list1=list(input)
list2=[]
i=0
ACount=0
CCount=0

k=True


#Gia tous anadiplasiasmous xrhsimopoisame counters wste na kalyptontai oi periorismoi.
#Periorismoi: poluA:mhkos 1 ews 5
#             poluC:mhkos 1 ews 10


while k:
    if list1[0]==list1[1] and list1[0]=='A' and ACount<5:
        del list1[1]
        ACount=ACount+1


    elif list1[0]==list1[1] and list1[0]=='A' and ACount==5:

        list2.append(list1[0])
        ACount = 0

```

```
while k:
    if list1[0]==list1[1] and list1[0]=='A' and ACount<5:
        del list1[1]
        ACount=ACount+1


    elif list1[0]==list1[1] and list1[0]=='A' and ACount==5:

        list2.append(list1[0])
        ACount = 0


    elif list1[0]==list1[1] and list1[0]=='C' and CCount<10:
        del list1[1]
        CCount=CCount+1

    elif list1[0]==list1[1] and list1[0]=='C' and CCount==10:

        list2.append(list1[0])
        CCount = 0


    elif list1[0] == list1[1] and list1[0] == 'G':
        del list1[1]


    elif list1[0] == list1[1] and list1[0] == 'T':
            del list1[1]
```

```

#diaforetiko gramma

    elif list1[0]!=list1[1]:
        list2.append(list1[0])
        del list1[0]
        ACount=0
        CCount=0



#telos tou while otan h lista exei 1 gramma kai prosthetetai sto telos ths listas mas.


    if len(list1)==1:
        list2.append(list1[0])
        del list1[0]
        k = False


print list2

a = ''.join(list2)

print ("H allhlouxia " +input+ "  apotelei molusmenh ekdosh ths " +a)
```

Εκτέλεση παραδείγματος βιβλίου:

```
C:\Users\Nikos\PycharmProjects\untitled1\venv\Scripts\python.exe "C:/Users/Ni
H allhlouxia AAATAAAGGGGCCCCCTTTTTTTCC  apotelei molusmenh ekdosh ths ATAGCTC

Process finished with exit code 0
```

Εκτέλεση παραδείγματος με τυχαία μολυσμένη αλληλουχία v= GGGCCCCCCCTTAAAAAAAAAACCCCCCCCCCCCTTGG:

```
C:\Users\Nikos\PycharmProjects\untitled1\venv\Scripts\python.exe "C:/Users/Nikos/Desktop/ΠΑΠΕΙ
H allhlouxia GGGCCCCCCCTTAAAAAAAAAACCCCCCCCCCCCTTGG  apotelei molusmenh ekdosh ths GCTAACCTG

Process finished with exit code 0
```

Άσκηση 11.4:

Στο σχήμα 11.7(σελ 451) φαίνεται ένα HMM με δύο καταστάσεις α και β. Όταν το HMM βρίσκεται στην κατάσταση α, έχει μεγαλύτερη πιθανότητα να εκπέμψει πουρίνες (A και G). Όταν βρίσκεται στην κατάσταση β, έχει μεγαλύτερη πιθανότητα να εκπέμψει πυριμιδίνες (C και T). Αποκωδικοποιήστε την πιο πιθανή ακολουθία των καταστάσεων (α/β) για την αλληλουχία GGCT. Χρησιμοποιήστε λογαριθμικές βαθμολογίες αντί για κανονικές βαθμολογίες πιθανοτήτων.

Λύση:

HMM A:

Πιθανότητα Αρχικής κατάστασης: 0.5

Πιθανότητα Επαναφοράς: 9/10 = 0.9

Πιθανότητα Αλλαγής Κατάστασης: 1/10 = 0.1

Πιθανότητα Επιλογής A: 2/5 = 0.4

Πιθανότητα Επιλογής G: 2/5 = 0.4

Πιθανότητα Επιλογής C: 1/10 = 0.1

Πιθανότητα Επιλογής T: 1/10 = 0.1

HMM B:

Πιθανότητα Αρχικής κατάστασης: 0.5

Πιθανότητα Επαναφοράς: 9/10 = 0.9

Πιθανότητα Αλλαγής Κατάστασης: 1/10 = 0.1

Πιθανότητα Επιλογής A: 1/5 = 0.2

Πιθανότητα Επιλογής G: 1/5 = 0.2

Πιθανότητα Επιλογής C: 3/10 = 0.3

Πιθανότητα Επιλογής T: 3/10 = 0.3

Δίνοντας την ακολουθία ***GGCT***

Αρχικοποιεί τις 2 καταστάσεις με πιθανότητα 0.5, έπειτα θα κοιτάξει προς το G κοιτώντας την λογαριθμικη πιθανότητα του HMM A να επιλέξει το G και την λογαριθμικη πιθανότητα του HMM B να επιλέξει το G. Επιλέγουμε την κατάσταση **α** με μεγαλύτερο σκορ (loge(1.2214027581601699)=0.2 > loge(1.1051709180756477) ==0.1).

Start > A

Έπειτα θα κοιτάξει προς το δεύτερο γράμμα G, κάνοντας επαναφορά(reset) την κατάσταση απο το κερδισμένο μονοπάτι(A στην περίπτωση μας) πολλαπλασιάζοντας με την λογαριθμικη πιθανότητα να επιλεγεί η κατάσταση A ή B.

Σκορ A : loge(1.0746553440638136) = 0.072

Σκορ B: loge(1.0040080106773419) = 0.004

Σκορ A > Σκορ B

Οπότε θα επιλεγεί η κατάσταση A

Start > A > A

Συνεχίζοντας με το C θα ξαναεπαναφέρουμε την κατάσταση απο το κερδισμένο μονοπάτι (A) πολλαπλασιάζοντας παλι με την λογαριθμικη πιθανότητα να επιλεγεί η επόμενη κατάσταση A ή B

Σκορ A: loge(1.0065010406231938) = 0.00648

Σκορ B: loge(1.0021623344805235) = 0.00216

Σκορ A > Σκορ B

Οποτε θα επιλεγεί η κατάσταση A

Start > A > A > A

Με το ίδιο τρόπο συνεχίζουμε στο T

Σκορ A: loge(1.0005833700941846) = 0.0005832

Σκορ B: loge(1.0001944188969045) = 0.0001944

Σκορ A > Σκορ B

Οπότε θα επιλεγεί η κατάσταση A

Start > A > A > A > A

Καλύτερο μονοπάτι για την αλληλουχία GGCT **: Start > A > A > A > A > End.**
Με σκορ loge(1.0005833700941846) = 0.0005831999999999.

Κώδικας:

Χρησιμοποιήσαμε τον constructor __init__ για την κλαση μας και την παραμετρο self για να δώσουμε ορίσματα στα αντικείμενα μας(καταστάσεις και πιθανότητες).

```
import numpy as np
import math


class ex11_4:

    def __init__(self, state, init, reset, change, a, g, t, c):
        self.state = state
        self.init = np.exp(init)
        self.reset_state = np.exp(reset)
        self.change_state = np.exp(change)
        self.goA = np.exp(a)
        self.goG = np.exp(g)
        self.goT = np.exp(t)
        self.goC = np.exp(c)

    def ___(self):
        return 'HMM: ' % self.name
```

```
Sequence = list('GGCT')
Seq_len = len(Sequence)

print('\n> Sequence: "%s"' % ''.join(Sequence))


HMM_A = exl1_4(state='A', init=0.5, reset=0.9, change=0.1, a=0.4, g=0.4, t=0.1, c=0.1)
HMM_B = exl1_4(state='B', init=0.5, reset=0.9, change=0.1, a=0.2, g=0.2, t=0.3, c=0.3)




# Arxikopoihsh ton score,me bash tis pithanothtes epiloghs kathe katastashs.

pathA = HMM_A.init
pathB = HMM_B.init
print('Path A: loge(%0.16f) = %0.8f' % (pathA, math.log(pathA)))
print('Path B: loge(%0.16f) = %0.8f' % (pathB, math.log(pathB)))
```

```
#krataei to path
path = ['Start']
#Arxikopoiei to kalutero score ws 0 gia thn periptwsh pou dwsoume adeia akolouthia
best_score = 0



#  Gia  na broume to megethos ths akolouthias.
for char in Sequence:

    print('\n> Looking for: %s.' % char)
```

```

    if char == 'G':
        pathA = np.exp((math.log(HMM_A.goG) * math.log(pathA)))
        pathB = np.exp((math.log(HMM_B.goG) * math.log(pathB)))

    elif char == 'C':
        pathA = np.exp((math.log(HMM_A.goC) * math.log(pathA)))
        pathB = np.exp((math.log(HMM_B.goC) * math.log(pathB)))

    elif char == 'T':
        pathA = np.exp((math.log(HMM_A.goT) * math.log(pathA)))
        pathB = np.exp((math.log(HMM_B.goT) * math.log(pathB)))

    else:
        pathA = np.exp((math.log(HMM_A.goA) * math.log(pathA)))
        pathB = np.exp((math.log(HMM_B.goA) * math.log(pathB)))

    print('Score A: loge(%0.16f) = (%0.8f).' % (pathA, math.log(pathA)))
    print('Score B: loge(%0.16f) = (%0.8f).' % (pathB, math.log(pathB)))

    best_score = max(pathA, pathB)
```

```
    if pathA >= pathB:
        pathB = pathA
        winner = 'A'
        path.append('A')
        #reset
        pathA = np.exp((math.log(HMM_A.reset_state) * math.log(pathA)))
        pathB = np.exp((math.log(HMM_B.change_state) * math.log(pathB)))

    else:
        pathA = pathB
        winner = 'B'
        path.append('B')
        #reset
        pathA = np.exp((math.log(HMM_A.change_state) * math.log(pathA)))
        pathB = np.exp((math.log(HMM_B.reset_state) * math.log(pathB)))

    print('Highest Score: Path %s.' % winner)
    print('Path:  %s.' % ' -> '.join(path))

print('\nBest Path: %s -> End.' % (' -> '.join(path)))
print('Scoring:   loge(%0.16f) = %0.16f.' % (best_score, math.log(best_score)))
```

Εκτέλεση παραδείγματος Βιβλίου με αλληλουχία GGCT:

```
C:\Users\Nikos\PycharmProjects\untitled\venv\Scripts\python.exe "C:/Users/Niko

> Sequence: "GGCT"
Path A: loge(1.6487212707001282) = 0.50000000
Path B: loge(1.6487212707001282) = 0.50000000

> Looking for: G.
Score A: loge(1.2214027581601699) = (0.20000000).
Score B: loge(1.1051709180756477) = (0.10000000).
Highest Score: Path A.
Path:  Start -> A.

> Looking for: G.
Score A: loge(1.0746553440638136) = (0.07200000).
Score B: loge(1.0040080106773419) = (0.00400000).
Highest Score: Path A.
Path:  Start -> A -> A.

> Looking for: C.
Score A: loge(1.0065010406231938) = (0.00648000).
Score B: loge(1.0021623344805235) = (0.00216000).
Highest Score: Path A.
Path:  Start -> A -> A -> A.

> Looking for: T.
Score A: loge(1.0005833700941846) = (0.00058320).
Score B: loge(1.0001944188969045) = (0.00019440).
Highest Score: Path A.
Path:  Start -> A -> A -> A -> A.

Best Path: Start -> A -> A -> A -> A -> End.
Scoring:   loge(1.0005833700941846) = 0.0005831999999999.

Process finished with exit code 0
```

Εκτέλεση παραδείγματος με τυχαία ακολουθία TATGGC:

```
C:\Users\Nikos\PycharmProjects\untitled\venv\Scripts\python.

> Sequence: "TATGGC"
Path A: loge(1.6487212707001282) = 0.50000000
Path B: loge(1.6487212707001282) = 0.50000000

> Looking for: T.
Score A: loge(1.0512710963760241) = (0.05000000).
Score B: loge(1.1618342427282831) = (0.15000000).
Highest Score: Path B.
Path:  Start -> B.

> Looking for: A.
Score A: loge(1.0060180360540649) = (0.00600000).
Score B: loge(1.0273678027634894) = (0.02700000).
Highest Score: Path B.
Path:  Start -> B -> B.

> Looking for: T.
Score A: loge(1.0002700364532808) = (0.00027000).
Score B: loge(1.0073166367379323) = (0.00729000).
Highest Score: Path B.
Path:  Start -> B -> B -> B.
```

```
> Looking for: G.
Score A: loge(1.0002916425194128) = (0.00029160).
Score B: loge(1.0013130613111163) = (0.00131220).
Highest Score: Path B.
Path:  Start -> B -> B -> B -> B.

> Looking for: G.
Score A: loge(1.0000524893775191) = (0.00005249).
Score B: loge(1.0002362238964715) = (0.00023620).
Highest Score: Path B.
Path:  Start -> B -> B -> B -> B -> B.

> Looking for: C.
Score A: loge(1.0000023619627894) = (0.00000236).
Score B: loge(1.0000637749535359) = (0.00006377).
Highest Score: Path B.
Path:  Start -> B -> B -> B -> B -> B -> B.

Best Path: Start -> B -> B -> B -> B -> B -> B -> End.
Scoring:   loge(1.0000637749535359) = 0.0000637729200000.

Process finished with exit code 0
```

Πηγές:

NEIL C. JONES ΚΑΙ PAVEL A. PEVZNER (2004), ΕΙΣΑΓΩΓΗ ΣΤΟΥΣ ΑΛΓΟΡΙΘΜΟΥΣ ΒΙΟΠΛΗΡΟΦΟΡΙΚΗΣ.

https://vlab.amrita.edu/?sub=3&brch=274&sim=1431&cnt=1

https://vlab.amrita.edu/index.php?sub=3&brch=274&sim=1433&cnt=1