Εργασία Βιοπληροφορικής

εαρινό εξάμηνο ακαδημαϊκού έτους 2023-2024

ΜΑΤΙΝΑ ΠΑΠΑΔΑΚΟΥ - Π21127

ΓΕΩΡΓΙΟΣ ΧΡΙΣΤΟΠΟΥΛΟΣ – Π20206

ΠΕΡΙΕΧΟΜΕΝΑ

ΘΕΜΑ i

(a) Αφού καλέσουμε τις βιβλιοθήκες random & sys , θα χρειαστούμε 2 μεθόδους για το πρώτο υπό-ερώτημα.

Η synthesize_string() επιλέγει έναν αριθμό από το ένα έως το τρία, καλεί την random_symbols(n) και αυτή επιστρέφει ένα string με n σύμβολα που διαλέγει τυχαία από την λίστα alphabet .

ask1.py

source2024 > ask1.py > synthesize_string

```python
import random
import sys

sys.stdout.reconfigure(encoding='utf-8') #Για να είμαστε σίγουροι ότι δεν θα μας δημιουργήσουν πρόβλημα τα σχόλια που είναι στα ελληνικά

def random_symbols(n):
    alphabet = ['A', 'C', 'G', 'T'] # Αλφάβητο
    return ''.join(random.choices(alphabet, k=n)) # Επιλέγουμε n τυχαία σύμβολα από το αλφάβητο

def synthesize_string():
    num_symbols = random.randint(1, 3) # Επιλέγουμε τυχαία 1 έως 3 σύμβολα
    symbols = random_symbols(num_symbols) # Παίρνουμε τα τυχαία σύμβολα
    return symbols
```

Τα τυχαία σύμβολα θα καταχωρηθούν στην συμβολοσειρά string(μέσα στην main_code_to_generate_a_string()) :

```python
string = synthesize_string() # erotima (i-a)
```

(b) Για κάθε ένα από τα 4 patterns θα εκτελεστεί το block κώδικα μέσα στην for. Η extend_string σπάει το pattern στα 3 μέρη ως εξής:
Έστω το pattern = AATTGA, αν επιλέξουμε τυχαία να αντικαταστήσουμε το γράμμα στην θέση 5, το G, τότε pattern[:x-1] = AATT και pattern[x:] = A και με την βοήθεια της random.choice([‘A’, ‘C’,’G’, ‘T’ , ‘ ‘]) επιλέγεται τυχαία ένα γράμμα ή το κενό. Άρα AATT + A + A = AATTAA.

Ο κώδικας που το επιτυγχάνει αυτό είναι:

```python
def main_code_to_generate_a_string():
    # Δοκιμή με τα δεδομένα patterns
    patterns = ["AATTGA", "CGCTTAT", "GGACTCAT", "TTATTCGTA"]

    string = synthesize_string() # erotima (i-a)

    version = 1
    for pattern in patterns: # erotima (i-b)
        num_symbols = random.randint(1, 2) # πόσα σύμβολα θα αντικαταστήσουμε(το πολύ 2)
        for i in range(1, num_symbols + 1):
            x = int(random.randint(1, len(pattern) - 1 ))
            choices = ['A', 'C', 'G', 'T', '']
            choices.remove(pattern[x])  #έτσι είμαστε σίγουροι ότι δεν θα αντικαταστήσει ένα γράμμα με τον εαυτό
            extend_string = pattern[:x-1] + random.choice(choices) + pattern[x:] # αντικατάσταση με ένα άλλο τυχ
            pattern = extend_string
        string += extend_string
        version += 1

    for i in range(1, len(patterns)): # erotima (i-c)
        string = string + random_symbols(random.randint(1, 2))
    return string
```

Ξεκινάμε αρχικοποιώντας την συμβολοσειρά string με την διαδικασία του ερωτήματος (1)

Στην συνέχεια για κάθε pattern από την λίστα patterns, επιλέγονται τυχαία πόσα σύμβολα (1ή2) από το pattern θα αντικατασταθούν. Για κάθε επιλεγμένο σύμβολο, ορίζεται μια τυχαία θέση στο pattern, όπου θα γίνει η αντικατάσταση.

Και έπειτα το σύμβολο στην συγκεκριμένη θέση αντικαθίσταται με ένα άλλο τυχαία επιλεγμένο σύμβολο από το αλφάβητο (A, C, G, T), είτε με κενό, ώστε να προκύψει μια τροποποιημένη έκδοση του pattern. Τελικά η νέα αυτή εκδοχή προστίθεται στην αρχική συμβολοσειρά string

Το δεύτερο for loop που φαίνεται στην παραπάνω στιγμιότυπο(γραμμή 39) εκτελεί το ερώτημα (c) που θα αναλυθεί παρακάτω.

(c) Τελειώνοντας το **ΘΕΜΑ Ι**, θα χρησιμοποιήσουμε πάλι την random_symbols(n), για να μας επιστρέψει ένα string με ένα έως δύο τυχαία σύμβολα.

```
39      for i in range(1, len(patterns)): # erotima (i-c)
40          string = string + random_symbols(random.randint(1, 2))
41      return string
```

Μένει να αναφέρουμε το πώς δημιουργήσαμε 50 συμβολοσειρές και τις καταχωρησαμε τυχαία σε 2 σύνολα datasetA[15] και datasetB[35]:

Καλούμε με επαναληπτική διαδικασία(for loop) την main_code_to_generate_a_string(), οπότε έχουμε την λίστα string με τις 50 συμβολοσειρές. Έπειτα ανοίγουμε(ή δημιουργούμε) το αρχείο κειμένου FullDataset και καταγράφουμε τα 50 strings με την βοήθεια της μεθόδου join(). Στη συνέχεια ανακατεύουμε την λίστα και αποθηκεύουμε στα datasetA & datasetB 15 και 35 συμβολοσειρές αντίστοιχα από την λίστα strings.

```
43  strings = []
44
45  for i in range(50):
46      strings.append(main_code_to_generate_a_string())
47
48  with open("auxiliary2024/FullDataset.txt", "w") as file:
49      file.write("\n".join(strings))
50      print(" - - Δημιουργήθηκε το αρχείο FullDataset - - ")
51
52  random.shuffle(strings) #ανακατευουμε την λίστα με τα 50 strings
53
54  datasetA = strings[:15]
55  datasetB = strings[15:]
56
57  print("DatasetA:", datasetA)
58  print("DatasetB:", datasetB)
```

Και εδώ αποθηκεύουμε στα αρχεία κειμένου datasetA, datasetB τις αντίστοιχες συμβολοσειρές

```
61  with open("auxiliary2024/datasetA.txt", "w", encoding="utf-8") as file:
62      file.write("\n".join(datasetA))
63      print(" - - Δημιουργήθηκε το αρχείο datasetA - - ")
64
65
66  with open("auxiliary2024/datasetB.txt", "w", encoding="utf-8") as file:
67      file.write("\n".join(datasetB))
68      print(" - - Δημιουργήθηκε το αρχείο datasetB - - ")
```

ΘΕΜΑ ii

- **Open DatasetA:**

  Αρχικά θα πρέπει να διαβάσουμε τα δεδομένα από το αρχείο κειμένου DatasetA με την βοήθεια της εντολής with open() as file, που θα εξασφαλίσει ότι το αρχείο θα κλείσει σωστά μετά την ανάγνωση. Συγκεκριμένα η μέθοδος readlines() θα διαβάσει όλες τις γραμμές του αρχείου καταχωρώντας το περιεχόμενο στην λίστα lines. Ενώ η μέθοδος strip() θα αφαιρέσει τυχόν περιττά κενά. Αν το αρχείο δεν βρεθεί ή αν προκύψει κάποιο άλλο σφάλμα όταν προσπαθήσουμε να διαβάσουμε το αρχείο, θα εκτυπωθεί ένα αντίστοιχο μήνυμα σφάλματος.

```
10  import numpy as np
11
12  file_name = 'auxiliary2024/datasetA.txt'
13
14  try:
15      with open(file_name, 'r') as file:
16          lines = file.readlines()
17          datasetA = [line.strip() for line in lines]
18
19  except FileNotFoundError:
20      print(f"The file {file_name} does not exist.")
21  except Exception as e:
22      print(f"An error: {e}")
```

- **global_alignment(A, B, alpha = 2):**

  Η μέθοδος αυτή στοχεύει να ευθυγραμμίσει δύο συμβολοσειρές σε όλο το μήκος τους, ακόμα και αν αυτό σημαίνει την εισαγωγή κενών για την βέλτιστη στοίχιση. Αναλυτικά:

  - Δημιουργούμε έναν πίνακα table για να αποθηκεύσουμε τις βέλτιστες τιμές κάθε στοίχισης για κάθε υπο=πρόβλημα. Τον αρχικοποιούμε με μηδενικά.
  - Alpha = 2 γιατί ΑΜ : 21127, 20206.
  - Για να υπολογίσουμε το σκορ έχουμε:
    Για κάθε ζεύγος χαρακτήρων (i, j) των δύο συμβολοσειρών, υπολογίζονται τρεις πιθανές βαθμολογίες
    - match: Προστίθεται στη βαθμολογία της διαγώνιας κίνησης. Προσθέτει 1 εάν οι χαρακτήρες είναι ίδιοι, διαφορετικά αφαιρεί alpha/2.

- delete: Προστίθεται στη βαθμολογία για κίνηση προς τα πάνω
- insert: Προστίθεται στη βαθμολογία για κίνηση προς τα αριστερά

Και η μέγιστη από αυτές τις τρεις τιμές αποθηκεύεται στο κελί του πίνακα table[i][j]

```python
def global_alignment(A, B, alpha=2):
    m, n = len(A), len(B)
    table = np.zeros((m + 1, n + 1))
    gap_penalty = -alpha
    match_score = 1
    mismatch_penalty = -alpha / 2

    # Initialization
    for i in range(1, m + 1):
        table[i][0] = i * gap_penalty
    for j in range(1, n + 1):
        table[0][j] = j * gap_penalty

    # Scoring
    for i in range(1, m + 1):
        for j in range(1, n + 1):
            match = table[i - 1][j - 1] + (match_score if A[i - 1] == B[j - 1] else mismatch_penalty)
            delete = table[i - 1][j] + gap_penalty
            insert = table[i][j - 1] + gap_penalty
            table[i][j] = max(match, delete, insert)
```

- Ξεκινώντας από το τελευταίο κελί του πίνακα, ακολουθείται η διαδρομή με τη μέγιστη τιμή προς τα πίσω για να βρούμε την βέλτιστη στοίχιση.

```python
    # Traceback
    align1, align2 = [], []
    i, j = m, n

    while i > 0 and j > 0:
        score_current = table[i][j]
        score_diagonal = table[i-1][j-1]
        score_left = table[i][j-1]
        score_up = table[i-1][j]

        max_score = max(score_diagonal, score_up, score_left)
        if max_score == score_diagonal:
            #print(" - -Diagonal- -")
            #print(score_diagonal)
            align1.append(A[i-1])
            align2.append(B[j-1])
            #print(align1, align2)
            i -= 1
            j -= 1
        elif max_score == score_left:
            #print(" - -LEFT- -")
            #print(score_left)
            align1.append("-")
            align2.append(B[j-1])
            #print(align1, align2)
            j -= 1
        elif max_score == score_up:
            #print(" - -UP- -")
            #print(score_up)
            align1.append(A[i-1])
            align2.append("-")
            #print(align1, align2)
            i -= 1
```

- Αν η διαδρομή ακολουθεί τη διαγώνια κίνηση, τότε οι χαρακτήρες ευθυγραμμίζονται. Ενώ εάν ακολουθείται η κίνηση προς τα πάνω ή αριστερά, εισάγεται κανό σε μια από τις δυο συμβολοσειρές.
- Και τέλος επιστρέφονται οι δύο ευθυγραμμισμένες συμβολοσειρές μαζί με την συνολική βαθμολογία της στοίχισης.

```
84      align1.reverse()
85      align2.reverse()
86      return ''.join(align1), ''.join(align2), int(table[m][n])
```

➢ **pairwise_distance_matrix(sequences, alpha=2):**

```
def pairwise_distance_matrix(sequences, alpha=2):
    n = len(sequences)
    dist_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(i + 1, n):
            _, _, score = global_alignment(sequences[i], sequences[j], alpha)
            dist_matrix[i][j] = dist_matrix[j][i] = score
    return dist_matrix
```

Η συνάρτηση αυτή η οποία υπολογίζει έναν **πίνακα αποστάσεων,** και συνεπώς υπολογίζει το σκορ, δηλαδή την συμβατότητα των αλληλουχιών. Ένα υψηλότερο σκορ δείχνει ότι οι αλληλουχίες είναι πιο όμοιες, ενώ ένα χαμηλότερο σκορ (περισσότερο αρνητικό) δείχνει ότι οι αλληλουχίες διαφέρουν περισσότερο.

- n = len(sequences): Υπολογίζει τον αριθμό των αλληλουχιών στη λίστα sequences.
- dist_matrix = np.zeros((n, n)): Δημιουργεί έναν πίνακα (μήτρα) n x n γεμάτο με μηδενικά, ο οποίος θα αποθηκεύει τις αποστάσεις μεταξύ των αλληλουχιών. Ο πίνακας αυτός θα είναι συμμετρικός, καθώς η απόσταση μιας αλληλουχίας με μια άλλη είναι η ίδια και στις δύο κατευθύνσεις (δηλαδή, η απόσταση από A προς B είναι ίση με την απόσταση από B προς A).
- Το διπλό for loop (για κάθε i και j) διατρέχει κάθε ζεύγος αλληλουχιών. Η πρώτη επανάληψη (με το i) επιλέγει μία αλληλουχία, και η δεύτερη επανάληψη (με το j) επιλέγει μία άλλη αλληλουχία για να την ευθυγραμμίσει με την πρώτη.
- Το διπλό for loop (για κάθε i και j) διατρέχει κάθε ζεύγος αλληλουχιών. Η πρώτη επανάληψη (με το i) επιλέγει μία αλληλουχία, και η δεύτερη επανάληψη (με το j) επιλέγει μία άλλη αλληλουχία για να την ευθυγραμμίσει με την πρώτη.
- _, _, score = global_alignment(sequences[i], sequences[j], alpha): Εδώ καλείται η συνάρτηση global_alignment, η οποία υπολογίζει το βέλτιστο ταίριασμα μεταξύ των αλληλουχιών sequences[i] και sequences[j]. Επιστρέφει τρία αποτελέσματα: την στοίχιση των δύο αλληλουχιών (align1, align2) και το τελικό σκορ ταύτισης (score), το οποίο δείχνει πόσο καλή είναι η στοίχιση μεταξύ των δύο αλληλουχιών.
- dist_matrix[i][j] = dist_matrix[j][i] = score: Η απόσταση μεταξύ των αλληλουχιών i και j αποθηκεύεται τόσο στη θέση (i, j) όσο και στη θέση (j, i) του πίνακα αποστάσεων. Αυτό γίνεται επειδή η απόσταση είναι συμμετρική (δηλαδή, δεν έχει σημασία ποια αλληλουχία είναι πρώτη ή δεύτερη).

- **neighbor_joining(dist_matrix):**

```
def neighbor_joining(dist_matrix):
    n = len(dist_matrix)
    clusters = [[i] for i in range(n)]
    while len(clusters) > 1:
        min_dist = float('inf')
        a, b = -1, -1
        for i in range(len(clusters)):
            for j in range(i + 1, len(clusters)):
                dist = -sum(dist_matrix[p][q] for p in clusters[i] for q in clusters[j]) / (len(clusters[i]) * len(clusters[j]))
                if dist < min_dist:
                    min_dist = dist
                    a, b = i, j
        new_cluster = clusters[a] + clusters[b]
        clusters.append(new_cluster)
        clusters.pop(max(a, b))
        clusters.pop(min(a, b))
    if len(clusters[0]) % 2 == 1:
        temp = clusters[0][0]
        clusters[0].pop(0)
        clusters[0].append(temp)

    return clusters[0]
```

Αυτός ο κώδικας υλοποιεί τον αλγόριθμο **Neighbor-Joining**, που χρησιμοποιείται για τη δημιουργία ενός **φυλογενετικού δέντρου** από έναν πίνακα αποστάσεων(scores). Ο αλγόριθμος αυτός ομαδοποιεί βήμα-βήμα τα πιο κοντινά μεταξύ τους clusters αλληλουχιών

- n = len(dist_matrix) υπολογίζει τον αριθμό των αλληλουχιών.
- clusters = [[i] for i in range(n)] δημιουργεί αρχικά έναν πίνακα όπου κάθε αλληλουχία είναι ένα ξεχωριστό cluster, με κάθε αλληλουχία να έχει τη δική της ομάδα (π.χ. [[0], [1], [2], ...]).
- Το while len(clusters) > 1 συνεχίζει μέχρι να ενωθούν όλες οι αλληλουχίες σε ένα σύμπλεγμα.
- Για κάθε ζευγάρι clusters (clusters i και j), υπολογίζεται η μέση απόσταση (δηλαδή το καλύτερο σκορ)
- Βρίσκεται το ζεύγος clusters με τη μικρότερη μέση απόσταση και αυτά τα δύο clusters ενώνονται σε ένα νέο cluster.
- Το while len(clusters) > 1 συνεχίζει μέχρι να ενωθούν όλες οι αλληλουχίες σε ένα cluster.
- Για κάθε ζευγάρι clusters (clusters i και j), υπολογίζεται η μέση απόσταση.
- Βρίσκεται το ζεύγος των clusters με τη μικρότερη μέση απόσταση και αυτά τα δύο clusters ενώνονται σε ένα νέο cluster.
- Για την διαχείριση του μονού αριθμού αλληλουχιών, δηλαδή αν το τελικό σύμπλεγμα έχει περιττό αριθμό αλληλουχιών, γίνεται εναλλαγή του πρώτου στοιχείου με το τελευταίο, διατηρώντας τη σειρά των αλληλουχιών.

➢ progressive_alignment(*sequences*):

Υλοποιεί τον αλγόριθμο **προοδευτικής στοίχισης (progressive alignment)** για την ευθυγράμμιση πολλών αλληλουχιών. Ο αλγόριθμος προοδευτικής στοίχισης ευθυγραμμίζει πρώτα τα ζεύγη αλληλουχιών που είναι πιο συμβατά μεταξύ τους βάση του σκορ τους και στη συνέχεια προχωρά στην ευθυγράμμιση των υπόλοιπων αλληλουχιών, ακολουθώντας μια ιεραρχία που καθορίζεται από ένα φυλογενετικό δέντρο.

```
def progressive_alignment(sequences):
    dist_matrix = pairwise_distance_matrix(sequences)
    guide_tree = neighbor_joining(dist_matrix)

    aligned_sequences = [sequences[guide_tree[0]], sequences[guide_tree[1]]]
    align1, align2, _ = global_alignment(aligned_sequences[0], aligned_sequences[1])
    aligned_sequences = [align1, align2]

    for idx in guide_tree[2:]:
        new_alignments = []
        for aligned_seq in aligned_sequences:
            align1, align2, _ = global_alignment(sequences[idx], aligned_seq)
            new_alignments.append(align2)
        new_alignments.insert(0, align1)
        aligned_sequences = new_alignments

    max_length = max(len(seq) for seq in aligned_sequences)
    aligned_sequences = [seq.ljust(max_length, '-') for seq in aligned_sequences]

    return aligned_sequences
```

- Ο πίνακας αποστάσεων (dist_matrix) υπολογίζει πόσο διαφέρουν οι αλληλουχίες μεταξύ τους και καθορίζει την συμβατότητα τους δημιουργώντας το αντίστοιχο σκορ.
- Χρησιμοποιείται η μέθοδος **Neighbor-Joining** για να δημιουργηθεί ένα φυλογενετικό δέντρο (guide_tree), που καθοδηγεί τη σειρά ευθυγράμμισης των αλληλουχιών.
- Οι δύο αλληλουχίες με το καλύτερο σκορ ευθυγραμμίζονται πρώτες με τη συνάρτηση global_alignment.
- Κάθε νέα αλληλουχία ευθυγραμμίζεται με τις ήδη ευθυγραμμισμένες ακολουθώντας το δέντρο.
- Όλες οι αλληλουχίες παίρνουν το ίδιο μήκος με κενά (-), για να ευθυγραμμιστούν σωστά.
- Τέλος επιστρέφονται οι τελικές, ευθυγραμμισμένες αλληλουχίες.

- **Αρχείο multiple_alignment_result.txt**

```
result = progressive_alignment(datasetA)
print("Multiple Sequence Alignment Result:")
for aligned_seq in result:
    print(aligned_seq)

print('')
try:
    output_file = 'auxiliary2024/multiple_alignment_result.txt'
    with open(output_file, 'w') as f:
        for aligned_seq in result:
            f.write(aligned_seq + '\n')

    print(f"Results written to {output_file}")
except FileNotFoundError:
    print(f"The file {file_name} does not exist.")
except Exception as e:
    print(f"An error occurred: {e}")
```

Με αυτόν τον τρόπο αποθηκεύονται, σε ένα αρχείο, οι αλληλουχίες που δημιουργήθηκαν κατά την πολλαπλή στοίχιση.

Παράδειγμα Εκτέλεσης:

```
Multiple Sequence Alignment Result:
GG-CAGTTCACGCTTT-CGA-T-CATTTGTTCGTATAAACT---
G--AAAACGAAGCTTATGGAAT-CATTTATTCGTAGACC-T---
GGGAATG--ACGCTTATGGA-CTCATTTATTAATACTT-CC---
ACCAATTG-ACACTTATGGTAT-CATTTGTTCGTACCC------
AAA--TT--G-AAGCGTATGGA-CTCATTTATTCGCATT--GGA
C-AAACTG--ACACTTGTGGA-CTCATTA-TTCGCATGT-GGC-
GTAA-T--CGACGCCTATGGA--CC-ATATATTCGCACGG--CA
AG-CAATTGACGATTGTG-A-CTCATTTAT-CC-T-AGG--AA-
AAAA-TTTA--CGGTTATGGA-CTCATTTATTCT-A--CCG-A-
AG-A-TT-G-ACGCCA-TTGA-CTCATTTATTG-TAG-CCTC--
AAA--T--G--ACC-TTATGA-CGCATATATTCGTAT--CACC-
T-A--TT---G-ACC-TTATGA-CTCATTCATTCGTAGGG-AA-
TCGA-CTTG-ACGCTGAT-GGA-T-CATTTACTCGTAGATG---
G-CG-TATTGACGATCATGGAC-ACATTCCTTCGTATT-CGAA-
C-GC-TT-GATGCTGAT-GCACTCATTTATTTGTAG-TTG----
```

ΘΕΜΑ iii

Δημιουργία HMM profile

- **check_region(align):**

```
def check_region(align):
    clean_align = align.replace("-", "")
    symbol_count = len(clean_align)
    conserved_regions = False
    deletions = False
    insertions = False

    # Calculate the percentage of letters (non-dash characters)
    if (symbol_count / len(align)) * 100 >= threshold:
        conserved_regions = True
        if symbol_count < len(align):
            deletions = True
    else:
        insertions = True

    return conserved_regions, deletions, insertions
```

Η πρώτη συνάρτηση που συναντάμε είναι η **is_conserved_region(align)**. Μετράει το πλήθος κάθε συμβόλου που υπάρχει σε κάθε στήλη της πολλαπλής στοίχισης και αν υπάρχει κάποιο σύμβολο που εμφανίζεται σε ένα συγκεκριμένο ποσοστό(π.χ. Threshold = 70%) επιστρέφει **True** και το κατατάσει ως match state, αν η ίδια στήλη έχει κάποιο '-' αλλά ικανοποιεί το threshold του 70% τότε το ορίζει ως delete state, αλλιώς ορίζεται ως insert state.

- **search_list(lst, element):**

```
def search_list(lst, element):
    i = 0
    while i < len(lst):
        if lst[i] == element:
            return True
        i += 1
    return False
```

Η συνάρτηση αυτή αναζητάει ένα στοιχείο μέσα σε μία λίστα.

- **take_the_column(n)**

```
def take_the_column(n):
    col = ""
    for i in range(len(multiple_alignment)):
        col = col + multiple_alignment[i][n]
    return col
```

Φτιάχνει μια συμβολοσειρά, την **col** όπου την επιστρέφει στο τέλος. Αυτή η συμβολοσειρά είναι μια στήλη του πίνακα **multiple_alignment**, ανάλογα με το n >=0 && n <= cols. Η κύρια χρήση της, είναι να ετοιμάσει την συμβολοσειρά για να την αξιοποιήσει η παραπάνω συνάρτηση και να ελέγξει αν είναι **conserved_region**.

- **hmm_profile(*multiple_alignment*):**

```
def hmm_profile(multiple_alignment):
    for i in range(num_cols):
        conserved, deletion, insertion = check_region(take_the_column(i))
        states.append(i)

        if (conserved): conserved_states.append(i)
        if (deletion): deletion_states.append(i)
        if (insertion): insertion_states.append(i)

    for i in range(num_cols):
        if not (search_list(deletion_states, states[i])) and search_list(conserved_states, states[i]) :
            hmmstates.append("match")
        elif search_list(deletion_states, states[i]) and search_list(conserved_states, states[i]) :
            hmmstates.append("delete")
        elif search_list(insertion_states, states[i] ):
            hmmstates.append("insert")
    print(conserved_states, deletion_states, insertion_states)
    print(hmmstates)

hmm_profile(multiple_alignment)
```

Η συνάρτηση hmm_profile(multiple_alignment) αναλύει μια πολλαπλή στοίχιση αλληλουχιών για να εντοπίσει διατηρημένες (conserved), διαγραμμένες (deleted), και εισαγμένες (inserted) περιοχές και να τις ταξινομήσει σε αντίστοιχες καταστάσεις (states) για τη δημιουργία ενός προφίλ κρυφού μοντέλου Markov (HMM).

- Αρχικά, υπολογίζεται ο αριθμός των στηλών του multiple_alignment και χρησιμοποιείται η λίστα states για να αποθηκεύσει κάθε στήλη της στοίχισης.
- Για κάθε στήλη στο multiple_alignment, η συνάρτηση καλεί τη βοηθητική συνάρτηση take_the_column(n) για να εξαγάγει την στήλη αυτή.
- Στη συνέχεια, καλεί τη συνάρτηση check_region(align) για να ελέγξει την κατάσταση της στήλης: αν είναι συντηρητική (conserved), αν έχει διαγραφές (deletions), ή αν έχει εισαγωγές (insertions).
- Αν η στήλη θεωρείται συντηρητική με βάση το threshold, η στήλη προστίθεται στη λίστα conserved_states.
- Αν η στήλη περιέχει διαγραφές (δηλαδή έχει πολλά κενά "gaps"), προστίθεται στη λίστα deletion_states.
- Αν υπάρχει εισαγωγή νέων χαρακτήρων σε αυτή τη θέση, προστίθεται στη λίστα insertion_states.
- Μετά την ανάλυση όλων των στηλών, η συνάρτηση δημιουργεί τη λίστα hmmstates, που αναπαριστά τον τύπο κατάστασης για κάθε στήλη (συντήρηση, διαγραφή, ή εισαγωγή):
    - "match": Αν η στήλη είναι συντηρητική (δηλαδή ανήκει στη λίστα conserved_states).
    - "delete": Αν η στήλη έχει διαγραφές και είναι επίσης συντηρητική.
    - "insert": Αν υπάρχουν εισαγωγές στην στήλη.

➢ create_Emmision_Prob_table():

Η συνάρτηση create_Emmision_Prob_table δημιουργεί έναν πίνακα εκπομπών για τις συντηρητικές στήλες μιας πολλαπλής στοίχισης αλληλουχιών. Ξεκινά αρχικοποιώντας έναν πίνακα με μηδενικές τιμές για τα σύμβολα A, C, T, G σε κάθε συντηρητική στήλη. Χρησιμοποιεί τον δείκτη conserved_index για να παρακολουθεί τη θέση των συντηρητικών στηλών και επεξεργάζεται κάθε στήλη. Αν η στήλη είναι συντηρητική (βρίσκεται στη λίστα conserved_states), εξάγει τη στήλη χωρίς τα κενά, μετρά την εμφάνιση των συμβόλων και υπολογίζει τις πιθανότητες εμφάνισής τους διαιρώντας τις εμφανίσεις κάθε συμβόλου με το συνολικό πλήθος συμβόλων της στήλης. Αυτές οι πιθανότητες στρογγυλοποιούνται και αποθηκεύονται στον πίνακα emission_table. Τέλος, εκτυπώνει τον πίνακα, δείχνοντας τις πιθανότητες για κάθε σύμβολο στις συντηρητικές στήλες.

```python
#Emmision Propability Table for HMM Profile
def create_Emmision_Prob_table():

    emission_table = {symbol: [0 for _ in range(len(conserved_states))] for symbol in alphabeta}

    conserved_index = 0
    for i in range(num_cols):
        if search_list(conserved_states, states[i]):
            col = take_the_column(i)
            col_no_gaps = col.replace("-", "")
            symbol_counts = Counter(col_no_gaps)
            total_symbols = len(col_no_gaps)

            for symbol in alphabeta:
                if symbol in symbol_counts:
                    prob = symbol_counts[symbol] / total_symbols
                else:
                    prob = 0
                emission_table[symbol][conserved_index] = round(prob, 3)
            conserved_index += 1

    print("\nEmission Probability Table:")
    for symbol in alphabeta:
        print(f"{symbol}: {emission_table[symbol]}")


create_Emmision_Prob_table()
```

- **create_Transition_Prob_table():**

```
def create_Transition_Prob_table():
    j = 0
    prob_table = [[0 for _ in range(len(conserved_states)-1)] for _ in range(9)]
    print("HMM Profile: ", hmmstates, "\n")
    count = [0 for _ in range(num_rows)]

    for i in range(len(hmmstates) - 1):

        if hmmstates[i] == "match" and hmmstates[i + 1] == "delete":
            count_letters = take_the_column(i+1)
            count_letters = count_letters.replace("-", "")
            count_letters = len(count_letters)/num_rows
            prob_table[0][j] = round(count_letters, 2)
            prob_table[2][j] = round(1 - count_letters, 2)
            j += 1
        elif hmmstates[i] == "match" and hmmstates[i + 1] == "insert":
            counter_of_inserts = 0
            prob_counter = 0
            i += 1

            while(hmmstates[i] == "insert"):
                inserts = take_the_column(i)

                for k in range(len(inserts)):
                    if inserts[k] == "-": count[k] += 1
                counter_of_inserts += 1
                i += 1

            for k in range(len(inserts)):
                if count[k] == 0: prob_counter += 1


            if counter_of_inserts > 2:
                prob_table[1][j] = round(prob_counter / num_rows, 2)
                prob_table[3][j] = round(prob_counter / counter_of_inserts, 2)
                prob_table[0][j] = round(1 - prob_table[1][j], 2)
                if(i+1 < len(hmmstates)) and  hmmstates[i + 1] != "delete":
                    prob_table[3][j] = round(prob_counter / counter_of_inserts, 2)
                    prob_table[4][j] = round(1 - prob_table[3][j], 2)
                if(i+1 < len(hmmstates)) and  hmmstates[i + 1] == "delete":
                    prob_table[4][j] = round((1 - prob_table[3][j])/2, 2)
                    prob_table[7][j] = round((1 - prob_table[3][j])/2, 2)
            else:
                prob_table[0][j] = round((1 - prob_table[3][j])/3, 2)
                prob_table[4][j] = round((1 - prob_table[3][j])/3, 2)
                prob_table[7][j] = round((1 - prob_table[3][j])/3, 2)
            j += 1
        elif hmmstates[i] == "match" and hmmstates[i + 1] == "match":
            prob_table[0][j] = 1
            j += 1
```

```
    elif hmmstates[i] == "delete" and hmmstates[i + 1] == "delete":
        count_letters = take_the_column(i+1)
        count_letters = count_letters.replace("-", "")
        count_letters = len(count_letters)/num_rows
        prob_table[0][j] = round(count_letters, 2)
        prob_table[5][j] = round(1 - count_letters, 2)
        if i + 2 < len(hmmstates) and hmmstates[i + 2] == "insert": prob_table[8][j] = round(prob_table[5][j], 2)
        j += 1
    elif hmmstates[i] == "delete" and hmmstates[i + 1] == "match":
        prob_table[6][j] = prob_table[2][j-1]
        prob_table[0][j] = prob_table[0][j-1]
        j += 1
    elif hmmstates[i] == "delete" and hmmstates[i + 1] == "insert":
        counter_of_inserts = 0
        prob_counter = 0

        while i + 1 < len(hmmstates) and hmmstates[i+1] == "insert":
            inserts = take_the_column(i)

            for k in range(len(inserts)):
                if inserts[k] == "-":
                    count[k] += 1
            counter_of_inserts += 1
            i += 1

        for k in range(len(inserts)):
            if count[k] == 0:
                prob_counter += 1

        if j < len(prob_table[0]):
            if counter_of_inserts > 0:
                if counter_of_inserts < prob_counter :prob_table[3][j] = round(counter_of_inserts / prob_counter, 2) #i -> i
                if counter_of_inserts >= prob_counter :prob_table[3][j] = round(prob_counter / counter_of_inserts, 2) #i -> i

                if j+1 < len(prob_table[0]):
                    prob_table[0][j+1] = round(prob_table[4][j], 2) #m -> m
                    prob_table[4][j] = round((1 - prob_table[3][j])/3, 2)  #i -> m
                    prob_table[7][j] = round(prob_table[4][j], 2)  #i -> d
            else:
                prob_table[7][j] = 0.5
                prob_table[4][j] = 0.5

        j += 1
```

```
    print("------Transition Probabilities------")
    for i in range(len(prob_table[0])):
        if prob_table[0][i] != 0: print("Transition: M" + str(i + 1) + " to M" + str(i + 2) + ": " + str(prob_table[0][i]))
        if prob_table[2][i] != 0: print("Transition: M" + str(i + 1) + " to D" + str(i + 1) + ": " + str(prob_table[2][i]))
        if prob_table[1][i] != 0: print("Transition: M" + str(i + 1) + " to I" + str(i + 1) + ": " + str(prob_table[1][i]))
        if prob_table[3][i] != 0: print("Transition: I" + str(i + 1) + " to I" + str(i + 1) + ": " + str(prob_table[3][i]))
        if prob_table[4][i] != 0: print("Transition: I" + str(i + 1) + " to M" + str(i + 2) + ": " + str(prob_table[4][i]))
        if prob_table[5][i] != 0: print("Transition: D" + str(i) + " to D" + str(i + 1) + ": " + str(prob_table[5][i]))
        if prob_table[6][i] != 0: print("Transition: D" + str(i) + " to M" + str(i + 2) + ": " + str(prob_table[6][i]))
        if prob_table[7][i] != 0: print("Transition: I" + str(i + 1) + " to D" + str(i + 2) + ": " + str(prob_table[7][i]))
        if prob_table[8][i] != 0: print("Transition: D" + str(i+1) + " to I" + str(i + 2) + ": " + str(prob_table[8][i]))

        print("--------------------------------")
    results = []
    for col in range(len(prob_table)):
        temp = 0
        for j in range(len(prob_table[col])):
            temp += prob_table[col][j]
        results.append(round(temp/len(prob_table[col]), 3))

    global transition_probs
    transition_probs = {
        'M': {'M': results[0], 'I': results[1], 'D': results[2]},
        'I': {'M': results[3], 'I': results[4], 'D': results[7]},
        'D': {'M': results[5], 'I': results[6], 'D': results[8]},
    }


create_Transition_Prob_table()
```

Αυτό το κομμάτι του κώδικα δημιουργεί έναν πίνακα πιθανότητας μετάβασης (Transition Probability Table) για ένα **Profile Hidden Markov Model (HMM)**. Ο στόχος του είναι να υπολογίσει τις πιθανότητες μετάβασης μεταξύ διαφορετικών καταστάσεων (Match, Insert, Delete) σε ένα HMM, βασιζόμενο στις καταστάσεις μιας αλληλουχίας.

Περιγραφή λειτουργίας:

1. **Αρχικοποίηση πίνακα πιθανότητας μετάβασης**:

   Η μεταβλητή prob_table είναι ένας πίνακας (2D list) που αρχικά γεμίζει με μηδενικά. Έχει 9 γραμμές, καθεμία από τις οποίες αντιστοιχεί σε μια συγκεκριμένη μετάβαση καταστάσεων(M -> M, M -> I, M -> D, I -> I, I -> M, D -> D, D -> M, I -> D, D -> I), και στήλες όσες οι διατηρημένες καταστάσεις (*conserved_states*).

   - Η λίστα count μετράει πόσες φορές συμβαίνουν τα κενά (gaps) κατά τις μεταβάσεις.

2. **Επεξεργασία των καταστάσεων**:

   - Ο αλγόριθμος επεξεργάζεται διαδοχικά τα στοιχεία της λίστας hmmstates (η οποία περιλαμβάνει τις καταστάσεις του μοντέλου όπως match, insert, delete) για να καθορίσει τι είδους μετάβαση γίνεται σε κάθε βήμα (π.χ. από match σε insert, από match σε delete, κλπ.).

   - Ανάλογα με το είδος της μετάβασης, υπολογίζει την αντίστοιχη πιθανότητα και την εισάγει στον πίνακα prob_table.

   - Σε περιπτώσεις όπου υπάρχει μετάβαση από insert ή delete σε άλλες καταστάσεις, γίνεται καταμέτρηση των εισαγωγών ή των κενού (gaps), και χρησιμοποιούνται για να υπολογιστούν οι πιθανότητες.

3. **Υπολογισμός πιθανοτήτων μετάβασης**:

   - Οι πιθανότητες μετάβασης υπολογίζονται με βάση τον αριθμό των παρατηρήσεων που συμβαίνουν σε κάθε μετάβαση καταστάσεων (π.χ., πόσες φορές γίνεται μετάβαση από match σε delete ή από insert σε match).

4. **Εκτύπωση πιθανών μεταβάσεων**:

   - Το πρόγραμμα εκτυπώνει όλες τις πιθανές μεταβάσεις από τη μια κατάσταση σε μια άλλη (π.χ., M1 -> M2, M1 -> D1, κλπ.) μαζί με τις αντίστοιχες πιθανότητες, αφού έχουν υπολογιστεί.

5. **Τελικός υπολογισμός και αποθήκευση των πιθανοτήτων**:

   - Μετά την επεξεργασία όλων των καταστάσεων και τη συμπλήρωση του πίνακα prob_table, ο αλγόριθμος υπολογίζει τον μέσο όρο των τιμών για κάθε πιθανότητα μετάβασης (M -> M, M -> I, κλπ.).

   - Αυτές οι πιθανότητες αποθηκεύονται σε ένα λεξικό transition_probs, το οποίο δομείται ως εξής:

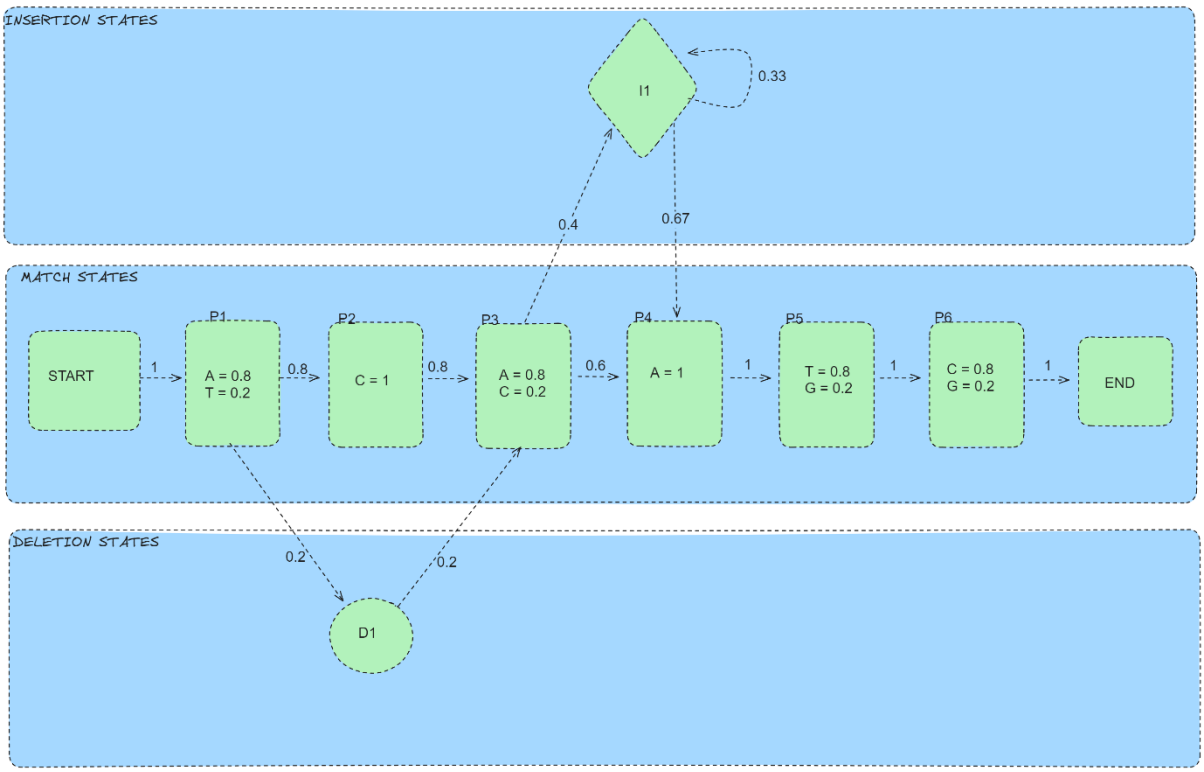
     - Για την κατάσταση M (Match), υπάρχουν πιθανότητες μετάβασης σε M, I (Insert), και D (Delete).

     - Αντίστοιχα, το ίδιο συμβαίνει για τις καταστάσεις I και D.

Ο τελικός πίνακας πιθανότητας μετάβασης μπορεί να χρησιμοποιηθεί για να περιγράψει πώς αλλάζουν οι καταστάσεις σε ένα μοντέλο HMM και χρησιμοποιείται σε αλγόριθμους όπως ο **Viterbi** για να βρει την πιο πιθανή αλληλουχία καταστάσεων.

Παράδειγμα απεικόνιση ενός HMM Profile με πολλαπλή στοίχιση:

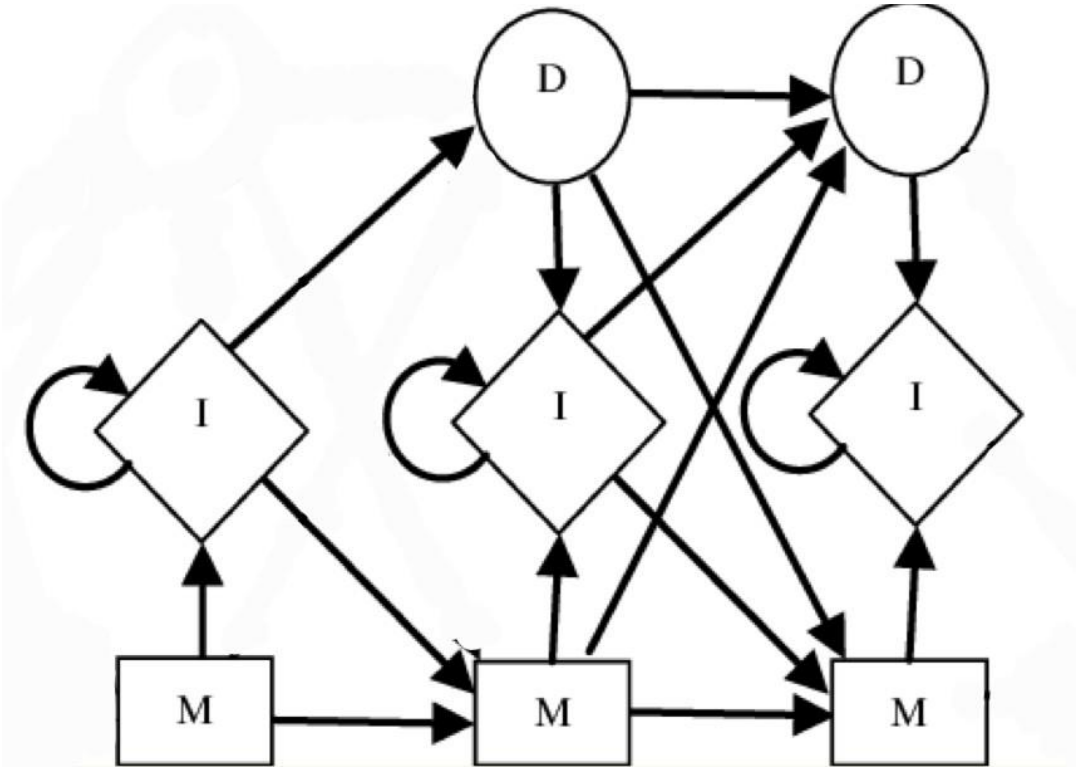
ACA---ATG
TCAACTATC
ACAC--AGC
ACA---ATC
A-C---ATC



Το αποτέλεσμα του κώδικά μας:

```
------Transition Probabilities------
Transition: M1 to M2: 0.8
Transition: M1 to D1: 0.2
-------------------------------
Transition: M2 to M3: 0.8
-------------------------------
Transition: M3 to M4: 0.6
Transition: M3 to I3: 0.4
Transition: I3 to I3: 0.33
Transition: I3 to M4: 0.67
Transition: D2 to M4: 0.2
-------------------------------
Transition: M4 to M5: 1
-------------------------------
Transition: M5 to M6: 1
-------------------------------
```

Γενική Μορφή του HMM Profile μας:



Παράδειγμα Εκτέλεσης Κώδικα:

```
Emission Probability Table:
A: [0.4, 0.333, 0.727, 0.909, 0.167, 0, 0, 0.727, 0.417, 0.267, 0, 0, 0.091, 0.357, 0.25, 0.308, 0.133, 0.429, 0.071, 0.182, 0, 0.636, 0.083, 0.308, 0, 0.533, 0.077, 0.429, 0, 0.077, 0, 0.692, 0.267, 0.5, 0.154]
C: [0.133, 0.333, 0.091, 0, 0.083, 0.077, 0.091, 0, 0.583, 0.2, 0.533, 0.357, 0.182, 0, 0, 0, 0.067, 0.071, 0.357, 0.182, 0.643, 0.182, 0.167, 0, 0, 0.067, 0, 0, 0.533, 0.077, 0.357, 0, 0.267, 0.071, 0.462]
T: [0.267, 0.167, 0.091, 0, 0.75, 0.692, 0, 0, 0, 0.067, 0.2, 0.5, 0.636, 0.5, 0.667, 0.231, 0.333, 0.071, 0.143, 0.455, 0.214, 0.091, 0.75, 0.692, 0.929, 0.4, 0.769, 0.5, 0.4, 0.385, 0.643, 0.077, 0.267, 0, 0.231]
G: [0.2, 0.167, 0.091, 0.091, 0, 0.231, 0.909, 0.273, 0, 0.467, 0.267, 0.143, 0.091, 0.143, 0.083, 0.462, 0.467, 0.429, 0.429, 0.182, 0.143, 0.091, 0, 0, 0.071, 0, 0.154, 0.071, 0.067, 0.462, 0, 0.231, 0.2, 0.429, 0.154]

HMM Profile:  ['match', 'insert', 'delete', 'delete', 'delete', 'insert', 'delete', 'delete', 'delete', 'delete', 'delete', 'match', 'match', 'delete', 'delete', 'delete', 'delete', 'delete', 'match', 'delete', 'delete', 'delete', 'insert', 'delete', 'delete', 'delete', 'delete', 'delete', 'match', 'delete', 'delete', 'match', 'delete', 'delete', 'delete', 'match', 'delete', 'delete', 'insert', 'insert', 'insert', 'insert', 'insert', 'insert', 'insert', 'insert']
```

```
------Transition Probabilities-----
Transition: M1 to M2: 0.33
Transition: I1 to M2: 0.33
Transition: I1 to D2: 0.33
--------------------------------
Transition: M2 to M3: 0.73
Transition: D1 to D2: 0.27
--------------------------------
Transition: M3 to M4: 0.73
Transition: D2 to D3: 0.27
Transition: D3 to I4: 0.27
--------------------------------
Transition: I4 to I4: 0.12
Transition: I4 to M5: 0.29
Transition: I4 to D5: 0.29
--------------------------------
Transition: M5 to M6: 0.87
Transition: D4 to D5: 0.13
--------------------------------
Transition: M6 to M7: 0.73
Transition: D5 to D6: 0.27
--------------------------------
Transition: M7 to M8: 0.73
Transition: D6 to D7: 0.27
--------------------------------
Transition: M8 to M9: 0.8
Transition: D7 to D8: 0.2
--------------------------------
Transition: M9 to M10: 0.8
--------------------------------
Transition: M10 to M11: 1
--------------------------------
Transition: M11 to M12: 0.93
Transition: M11 to D11: 0.07
--------------------------------
Transition: M12 to M13: 0.73
Transition: D11 to D12: 0.27
--------------------------------
Transition: M13 to M14: 0.93
Transition: D12 to D13: 0.07
--------------------------------
Transition: M14 to M15: 0.8
Transition: D13 to D14: 0.2
--------------------------------
```

```
--------------------------------
Transition: M15 to M16: 0.87
Transition: D14 to D15: 0.13
--------------------------------
Transition: M16 to M17: 0.87
--------------------------------
Transition: M17 to M18: 0.93
Transition: M17 to D17: 0.07
--------------------------------
Transition: M18 to M19: 0.93
Transition: D17 to D18: 0.07
--------------------------------
Transition: M19 to M20: 0.73
Transition: D18 to D19: 0.27
Transition: D19 to I20: 0.27
--------------------------------
Transition: I20 to I20: 0.17
Transition: I20 to M21: 0.28
Transition: I20 to D21: 0.28
--------------------------------
Transition: M21 to M22: 0.73
Transition: D20 to D21: 0.27
--------------------------------
Transition: M22 to M23: 0.8
Transition: D21 to D22: 0.2
--------------------------------
Transition: M23 to M24: 0.87
Transition: D22 to D23: 0.13
--------------------------------
Transition: M24 to M25: 0.93
Transition: D23 to D24: 0.07
--------------------------------
Transition: M25 to M26: 0.93
--------------------------------
Transition: M26 to M27: 0.87
Transition: M26 to D26: 0.13
--------------------------------
Transition: M27 to M28: 0.93
Transition: D26 to D27: 0.07
--------------------------------
Transition: M28 to M29: 0.93
--------------------------------
Transition: M29 to M30: 0.87
Transition: M29 to D29: 0.13
--------------------------------
```

```
Transition: M31 to M32: 0.87
Transition: D30 to D31: 0.13
--------------------------------
Transition: M32 to M33: 0.87
--------------------------------
Transition: M33 to M34: 0.93
Transition: M33 to D33: 0.07
--------------------------------
Transition: M34 to M35: 0.87
Transition: D33 to D34: 0.13
Transition: D34 to I35: 0.13
--------------------------------
Emission Probabilities for Viterbi:
 {'M': [{'A': 0.23, 'C': 0.16, 'G': 0.18, 'T': 0.31}], 'I': [{'A': 0.26, 'C': 0.16, 'G': 0.22, 'T': 0.36}], 'D': [{'A': 0.27, 'C': 0.2, 'G': 0.27, 'T': 0.26}]}

Emission Probabilities for Viterbi:
 {'M': {'M': 0.787, 'I': 0.0, 'D': 0.014}, 'I': {'M': 0.009, 'I': 0.026, 'D': 0.026}, 'D': {'M': 0.103, 'I': 0.0, 'D': 0.02}}
```

Αλγόριθμος Viterbi

- **create_Transition_Prob_table():**

```python
def Emmisions_Prob_for_Viterbi():
    emission_probs = {'M': [], 'I': [], 'D': []}

    total_A, total_C, total_T, total_G = 0, 0, 0, 0

    for col in conserved_states:
        for row in multiple_alignment:
            if row[col] == 'A': total_A += 1
            if row[col] == 'C': total_C += 1
            if row[col] == 'T': total_T += 1
            if row[col] == 'G': total_G += 1

    total = (len(conserved_states))*num_rows - 1

    overall_prob_dict = {
        'A': round(total_A / total, 2),
        'C': round(total_C / total, 2),
        'G': round(total_G / total, 2),
        'T': round(total_T / total, 2)
    }

    emission_probs['M'].append(overall_prob_dict)

    total_A, total_C, total_T, total_G = 0, 0, 0, 0

    for col in insertion_states:
        for row in multiple_alignment:
            if row[col] == 'A': total_A += 1
            if row[col] == 'C': total_C += 1
            if row[col] == 'T': total_T += 1
            if row[col] == 'G': total_G += 1
```

```
total = total_A + total_C + total_T + total_G

if total > 0:
    overall_prob_dict = {
        'A':  round(total_A / total, 2),
        'C':  round(total_C / total, 2),
        'G':  round(total_G / total, 2),
        'T':  round(total_T / total, 2)
    }
else:
    overall_prob_dict = {
        'A': 0.0,
        'C': 0.0,
        'G': 0.0,
        'T': 0.0
    }

emission_probs['I'].append(overall_prob_dict)
```

```
total_A, total_C, total_T, total_G, total = 0, 0, 0, 0, 0
for col in deletion_states:
    for row_idx, row in enumerate(multiple_alignment):
        if row[col] == '-':
            total += 1
            back_col = col
            while back_col >= 0 and hmmstates[back_col] != 'match':
                back_col -= 1
            if back_col >= 0:
                match_nucleotide = multiple_alignment[row_idx][back_col]
                if match_nucleotide == 'A': total_A += 1
                if match_nucleotide == 'C': total_C += 1
                if match_nucleotide == 'T': total_T += 1
                if match_nucleotide == 'G': total_G += 1
```

```
    if total > 0:
        overall_prob_dict = {
            'A':  round(total_A / total, 2),
            'C':  round(total_C / total, 2),
            'G':  round(total_G / total, 2),
            'T':  round(total_T / total, 2)
        }
    else:
        overall_prob_dict = {
            'A': 0.0,
            'C': 0.0,
            'G': 0.0,
            'T': 0.0
        }
    emission_probs['D'].append(overall_prob_dict)

    return emission_probs

emission_probs = Emmisions_Prob_for_Viterbi()
```

Η συνάρτηση Emmisions_Prob_for_Viterbi() υπολογίζει τις πιθανότητες εκπομπής (emission probabilities) για τις καταστάσεις "M" (match), "I" (insert) και "D" (delete) ενός Κρυφού Μοντέλου Markov (HMM). Αυτές οι πιθανότητες αντιπροσωπεύουν την πιθανότητα εμφάνισης ενός συγκεκριμένου νουκλεοτιδίου (A, C, G, T) όταν το HMM βρίσκεται σε μία από αυτές τις καταστάσεις. Αρχικά, η συνάρτηση δημιουργεί ένα λεξικό emission_probs με κενές λίστες για κάθε κατάσταση του HMM. Στη συνέχεια, για την κατάσταση "M" (match), διατρέχει τις συντηρητικές στήλες της πολλαπλής ευθυγράμμισης (multiple_alignment) και μετρά πόσες φορές εμφανίζεται κάθε νουκλεοτίδιο στις συγκεκριμένες θέσεις. Από αυτές τις μετρήσεις, υπολογίζει τις πιθανότητες εμφάνισης κάθε νουκλεοτιδίου και τις αποθηκεύει για την κατάσταση "M". Ακολουθώντας την ίδια διαδικασία, υπολογίζει τις πιθανότητες εκπομπής για την κατάσταση "I" (insert), εξετάζοντας τις θέσεις εισαγωγών. Αν δεν υπάρχουν δεδομένα σε κάποια στήλη, οι πιθανότητες ορίζονται ως μηδενικές. Για την κατάσταση "D" (delete), η συνάρτηση δεν εξετάζει άμεσα τις διαγραφές, καθώς αυτές είναι κενά (-). Αντίθετα, εξετάζει το νουκλεοτίδιο που υπήρχε στην τελευταία θέση match πριν από το κενό και υπολογίζει τις πιθανότητες εμφάνισης για κάθε νουκλεοτίδιο πριν από μια διαγραφή. Τέλος, η συνάρτηση επιστρέφει το λεξικό emission_probs με τις πιθανότητες εκπομπής για τις καταστάσεις "M", "I" και "D", οι οποίες χρησιμοποιούνται στον αλγόριθμο Viterbi για τον υπολογισμό της πιο πιθανής αλληλουχίας καταστάσεων δεδομένων των παρατηρήσεων.