AKSHAI D PILLAI

AM.EN.U4AIE22102

Question_01

```
def HammingDistance(pattern1,pattern2):
  d=0
  if len(pattern1)==len(pattern2):
    for i in range(len(pattern1)):
      if pattern1[i]!=pattern2[i]:
        d+=1
  return d

def DistanceBetweenPatternAndStrings(pattern, dna):
    k = len(pattern)
    distance = 0
    for text in dna:
        min_distance = float('inf')
        for i in range(len(text) - k + 1):
            pattern_in_text = text[i:i+k]
            current_distance = HammingDistance(pattern, pattern_in_text)
            min_distance = min(min_distance, current_distance)
        distance += min_distance
    return distance

pattern = "AAA"
dna = ["TTACCTTAAC", "GATATCTGTC", "ACGGCGTTCG", "CCCTAAAGAG", "CGTCAGAGGT"]
print(DistanceBetweenPatternAndStrings(pattern, dna))
```

```
5
```

Question_02

```
def HammingDistance(pattern1,pattern2):
  d=0
  if len(pattern1)==len(pattern2):
    for i in range(len(pattern1)):
      if pattern1[i]!=pattern2[i]:
        d+=1
  return d

def GenerateKmerVariants(kmer, d):
    if d == 0:
        return [kmer]
    if len(kmer) == 1:
        return ['A', 'C', 'G', 'T']

    variants = []
    suffix_variants = GenerateKmerVariants(kmer[1:], d)
    for suffix in suffix_variants:
        if HammingDistance(kmer[1:], suffix) < d:
            for nucleotide in ['A', 'C', 'G', 'T']:
                variants.append(nucleotide + suffix)
        else:
            variants.append(kmer[0] + suffix)
    return variants

def MotifEnumeration(Dna, k, d):
    motifs = set()
    for i in range(len(Dna[0]) - k + 1):
        kmer = Dna[0][i:i+k]
        variants = GenerateKmerVariants(kmer, d)
        for variant in variants:
            if all(any(HammingDistance(variant, Dna_string[j:j+k]) <= d for j in range(len(Dna_string) - k + 1)) for Dna_string in Dna):
                motifs.add(variant)
    return motifs

k = 3
d = 1
Dna = [
    "ATTTGGC",
    "TGCCTTA",
    "CGGTATC",
    "GAAAATT"
]

motifs = MotifEnumeration(Dna, k, d)

for motif in motifs:
    print(motif, end=" ")
```

```
 ATA GTT TTT ATT
```

Question_03

```
def count_matrix(motifs):
    count = {'A': [], 'C': [], 'G': [], 'T': []}
    for i in range(len(motifs[0])):
        for nucleotide in count:
            count[nucleotide].append(sum(1 for motif in motifs if motif[i] == nucleotide))
    return count

def profile_matrix(count, t):
    profile = {'A': [], 'C': [], 'G': [], 'T': []}
    for nucleotide in count:
        for count_value in count[nucleotide]:
            profile[nucleotide].append(count_value / t)
    return profile

def consensus_string(profile):
    consensus = ""
    for i in range(len(profile['A'])):
        max_probability = 0
        max_nucleotide = ''
        for nucleotide in profile:
            if profile[nucleotide][i] > max_probability:
                max_probability = profile[nucleotide][i]
                max_nucleotide = nucleotide
        consensus += max_nucleotide
    return consensus

motifs = [
    "TCGGGGGTTTTT",
    "CCGGTGACTTAC",
    "ACGGGGATTTTC",
    "TTGGGGACTTTT",
    "AAGGGGACTTCC",
    "TTGGGGACTTCC",
    "TCGGGGATTCAT",
    "TCGGGGATTCCT",
    "TAGGGGAACTAC",
    "TCGGGTATAACC"
]

t = len(motifs)

count = count_matrix(motifs)

profile = profile_matrix(count, t)

consensus = consensus_string(profile)

print("Motif Matrix:")
for motif in motifs:
    print(motif)
print("\nCount Matrix:")
for nucleotide in count:
    print("{}: {}".format(nucleotide, " ".join(map(str, count[nucleotide]))))
print("\nProfile Matrix:")
for nucleotide in profile:
    print("{}: {}".format(nucleotide, " ".join(map(str, profile[nucleotide]))))
print("\nConsensus string:", consensus)
```

```
Motif Matrix:
TCGGGGGTTTTT
CCGGTGACTTAC
ACGGGGATTTTC
TTGGGGACTTTT
AAGGGGACTTCC
TTGGGGACTTCC
TCGGGGATTCAT
TCGGGGATTCCT
TAGGGGAACTAC
TCGGGTATAACC

Count Matrix:
A: 2 2 0 0 0 0 9 1 1 1 3 0
C: 1 6 0 0 0 0 0 4 1 2 4 6
G: 0 0 10 10 9 9 1 0 0 0 0 0
T: 7 2 0 0 1 1 0 5 8 7 3 4

Profile Matrix:
A: 0.2 0.2 0.0 0.0 0.0 0.0 0.9 0.1 0.1 0.1 0.3 0.0
```

```
C: 0.1 0.6 0.0 0.0 0.0 0.0 0.0 0.4 0.1 0.2 0.4 0.6
G: 0.0 0.0 1.0 1.0 0.9 0.9 0.1 0.0 0.0 0.0 0.0 0.0
T: 0.7 0.2 0.0 0.0 0.1 0.1 0.0 0.5 0.8 0.7 0.3 0.4

Consensus string: TCGGGGATTTCC
```

Question_04

```
def probability_of_kmer(kmer, profile):
    prob = 1
    for i, nucleotide in enumerate(kmer):
        if nucleotide == 'A':
            prob *= profile[0][i]
        elif nucleotide == 'C':
            prob *= profile[1][i]
        elif nucleotide == 'G':
```