Problem 1 : Compute Distance Between Pattern And Strings

Find the distance between a pattern and a set of strings.

Given: A DNA string Pattern and a collection of DNA strings *Dna*.

Return: *DistanceBetweenPatternAndStrings(Pattern, Dna)*.

```
DistanceBetweenPatternAndStrings(Pattern, Dna)
    k ← |Pattern|
    distance ← 0
    for each string Text in Dna
        HammingDistance ← ∞
        for each k-mer Pattern' in Text
            if HammingDistance > HammingDistance(Pattern, Pattern')
                HammingDistance ← HammingDistance(Pattern, Pattern')
        distance ← distance + HammingDistance
    return distance
```

Sample Dataset

AAA

TTACCTTAAC GATATCTGTC ACGGCGTTCG CCCTAAAGAG CGTCAGAGGT

Sample Output

5

```
def HammingDistance(text1, text2):
  d=0
  for i in range(len(text1)):
    if text1[i] != text2[i]:
      d+=1
  return d


def DistanceBetweenPatternAndStrings(Pattern, DNA):
  dist = 0
  k = len(Pattern)
  for dna_str in DNA:
    kmerdist = {dna_str[i:i+k]:HammingDistance(Pattern, dna_str[i:i+k]) for i in range(len(dna_str)-k+1)}
    dist += min(kmerdist.values())
  return dist

print(DistanceBetweenPatternAndStrings('AAA', ['TTACCTTAAC', 'GATATCTGTC', 'ACGGCGTTCG', 'CCCTAAAGAG', 'CGTCAGAGGT']))
```

```
5
```

Problem 2 : Brute Force Motif Search - Implanted Motif Problem

Implement Brute Force Motif Search for a set of DNA strings.

Given a collection of strings Dna and an integer d, a k-mer is a **(k,d)-motif** if it appears in every string from Dna with at most d mismatches. The following algorithm finds (k,d)-motifs.

```
MOTIFENUMERATION(Dna, k, d)
    Patterns ← an empty set
    for each k-mer Pattern in Dna
      for each k-mer Pattern' differing from Pattern by at most d
        mismatches
        if Pattern' appears in each string from Dna with at most d
        mismatches
          add Pattern' to Patterns
    remove duplicates from Patterns
    return Patterns
```

Implanted Motif Problem

Implement MotifEnumeration (shown above) to find all *(k, d)-motifs* in a collection of strings.

Given: Integers *k and d*, followed by a collection of strings *Dna*.

Return: All *(k, d)-motifs* in Dna.

Sample Dataset

```
3 1
ATTTGGC
TGCCTTA
CGGTATC
GAAAATT
```

Sample Output

```
ATA ATT GTT TTT
```

```
def generate_kmers(k):
  kmers = ['']
  bases = ['A', 'G', 'C', 'T']
  if k == 1:
    return bases
  for _ in range(k):
    imer = []
    for kmer in kmers:
      for base in bases:
        imer.append(kmer + base)
    kmers = imer
  return kmers


def MotifEnumeration(dna, k, d):
    patterns = set()
    for i in range(len(dna[0]) - k + 1):
        kmer = dna[0][i:i+k]
        neighbors = generate_kmers(k)
        for neighbor in neighbors:
            if sum([HammingDistance(neighbor, dna_str[i:i+k]) <= d for dna_str in dna]) == len(dna):
                patterns.add(neighbor)
    return patterns

dna = ['ATTTGGC', 'TGCCTTA', 'CGGTATC', 'GAAAATT']
k = 3
d = 1
motifs = MotifEnumeration(dna, k, d)
print("Motifs found:", motifs)
```

```
Motifs found: set()
```

Problem 3: Scoring Motifs

Given a set of '*t*' DNA Strings, display a Motif Matrix and calculate the corresponding Count matrix and Profile matrix. Use the profile matrix to form the Consensus string.

Dataset : Use NF-xB binding sites and form consensus "**TCGGGGATTTCC**"

1	T	C	G	G	G	G	g	T	T	T	t	t
2	c	C	G	G	t	G	A	c	T	T	a	C
3	a	C	G	G	G	G	A	T	T	T	t	C
4	T	t	G	G	G	G	A	c	T	T	t	t
5	a	a	G	G	G	G	A	c	T	T	C	C
6	T	t	G	G	G	G	A	c	T	T	C	C
7	T	C	G	G	G	G	A	T	T	c	a	t
8	T	C	G	G	G	G	A	T	T	c	C	t
9	T	a	G	G	G	G	A	a	c	T	a	C
10	T	C	G	G	G	t	A	T	a	a	C	C

```
def count(motifs):
    k = len(motifs[0])
    counts = {nucleotide: [0] * k for nucleotide in ['A', 'C', 'G', 'T']}
    for i in range(k):
        for motif in motifs:
            counts[motif[i]][i] += 1
    return counts

def profile(motifs):
    k = len(motifs[0])
    counts = count(motifs)
    return {nucleotide: [count / len(motifs) for count in counts[nucleotide]] for nucleotide in ['A', 'C', 'G', 'T']}

def consensus(profile):
    k = len(profile['A'])
    return ''.join(max(profile, key=lambda nucleotide: profile[nucleotide][i]) for i in range(k))

motifs = ["TCGGGGGTTTTT",
        "CCGGTGACTTAC",
        "ACGGGGATTTTC",
        "TTGGGGACTTTT",
        "AAGGGGACTTCC",
        "TTGGGGACTTCC",
        "TCGGGGATTCAT",
        "TCGGGGATTCCT",
        "TAGGGGAACTAC",
        "TCGGGTATAACC"]
t = len(motifs)
counts = count(motifs)
profile = profile(motifs)
consensus_string = consensus(profile)

print("Motif Matrix:")
for motif in motifs:
    print(motif)
print("\nCount Matrix:")
for nucleotide in ['A', 'C', 'G', 'T']:
    print(f"{nucleotide}: {' '.join(map(str, counts[nucleotide]))}")
print("\nProfile Matrix:")
for nucleotide in ['A', 'C', 'G', 'T']:
    print(f"{nucleotide}: {' '.join(map(str, profile[nucleotide]))}")
print("\nConsensus String:", consensus_string)
```

```
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
```

Problem 4: Find a Profile-most Probable k-mer in a String

Given a profile matrix *Profile*, we can evaluate the probability of every *k*-mer in a string *Text* and find a ***Profile*-most probable** *k*-mer in *Text*, i.e., a *k*-mer that was most likely to have been generated by *Profile* among all *k*-mers in *Text*.

***Profile*-most Probable *k*-mer Problem**

Find a Profile-most probable k-mer in a string.

Given: A string *Text*, an integer *k*, and a 4 × *k* matrix *Profile*.

Return: A *Profile*-most probable *k*-mer in *Text*. (If multiple answers exist, you may return any one.)

Sample Dataset

```
ACCTGTTTATTGCCTAAGTTCCGAACAAACCCAATATAGCCCGAGGGCCT
5
0.2 0.2 0.3 0.2 0.3
0.4 0.3 0.1 0.5 0.1
0.3 0.3 0.5 0.2 0.4
0.1 0.2 0.1 0.1 0.2
```

Sample Output