In [21]:

```python
# Task 1: Evolution as a Sequence of Mistakes

def HammingDistance(seq1, seq2):
  """Calculates the Hamming distance between two DNA sequences.

  Args:
    seq1: A DNA sequence.
    seq2: A DNA sequence of the same length as seq1.

  Returns:
    The Hamming distance between seq1 and seq2.
  """

  if len(seq1) != len(seq2):
    raise ValueError("Sequences must be of equal length.")

  distance = 0
  for i in range(len(seq1)):
    if seq1[i] != seq2[i]:
      distance += 1

  return distance

# Get input:
seq1 = input("Enter the first DNA sequence: ")
seq2 = input("Enter the second DNA sequence: ")

# Calculate and print the Hamming distance:
hamming_dist = HammingDistance(seq1, seq2)
print("The Hamming distance between the two sequences is:", hamming_dist)
```

```
Enter the first DNA sequence: GACTCGGA
Enter the second DNA sequence: CGATCGAC
The Hamming distance between the two sequences is: 5
```

In [3]:

```python
# Task 2: Translating RNA into Protein

def Translation(s: str) -> str:
    """Translates an RNA string into a protein string.

    Args:
        s: An RNA string.

    Returns:
        The protein string encoded by s.
    """
    CODON_TABLE = {
    'AUG': 'M', 'UUU': 'F', 'UUC': 'F', 'UUA': 'L', 'UUG': 'L', 'UCU': 'S',
    'UCC': 'S', 'UCA': 'S', 'UCG': 'S', 'UAU': 'Y', 'UAC': 'Y', 'UGU': 'C',
    'UGC': 'C', 'UGG': 'W', 'CUU': 'L', 'CUC': 'L', 'CUA': 'L', 'CUG': 'L',
    'CCU': 'P', 'CCC': 'P', 'CCA': 'P', 'CCG': 'P', 'CAU': 'H', 'CAC': 'H',
    'CAA': 'Q', 'CAG': 'Q', 'CGU': 'R', 'CGC': 'R', 'CGA': 'R', 'CGG': 'R',
    'AUU': 'I', 'AUC': 'I', 'AUA': 'I', 'ACU': 'T', 'ACC': 'T', 'ACA': 'T',
    'ACG': 'T', 'AAU': 'N', 'AAC': 'N', 'AAA': 'K', 'AAG': 'K', 'AGU': 'S',
    'AGC': 'S', 'AGA': 'R', 'AGG': 'R', 'GUU': 'V', 'GUC': 'V', 'GUA': 'V',
    'GUG': 'V', 'GCU': 'A', 'GCC': 'A', 'GCA': 'A', 'GCG': 'A', 'GAU': 'D',
    'GAC': 'D', 'GAA': 'E', 'GAG': 'E', 'GGU': 'G', 'GGC': 'G', 'GGA': 'G',
    'GGG': 'G', 'UAA': 'Stop', 'UAG': 'Stop', 'UGA': 'Stop'
    }
    protein = ""
    codon = ""

    for nucleotide in s: # loops through each nucleotide
        codon += nucleotide # Builds a codon by adding one nucleotide at a time

        if len(codon) == 3:
            amino_acid = CODON_TABLE.get(codon, None) # Looks up the codon in CODON_TABLE to find its corr
            if amino_acid == "Stop": # handle different cases
                break
            elif amino_acid is not None:
                protein += amino_acid
            codon = ""

    return protein

# Get input RNA string:
RNA_string = input("Enter an RNA string: ")
```

```
# Translate the RNA string into a protein string:
protein_string = Translation(RNA_string)

# Print the resulting protein string:
print("The corresponding protein string is:", protein_string)
```

```
Enter an RNA string: AGCTCGA
The corresponding protein string is: S
```

In [4]:

```python
# Task 3: Finding a Motif in DNA

def FindingMotif(s: str, t: str) -> list[int]:
    """Finds all locations of a substring t in a string s.

    Args:
        s: The string to search in.
        t: The substring to search for.

    Returns:
        A list of all locations of t in s.
    """

    locations = [] # Creates an empty list
    n = len(s) # Stores the length of the main and substring
    m = len(t)

    for i in range(n - m + 1):
        if s[i:i+m] == t:
            locations.append(i + 1) # If a match is found, adds position to the locations list.

    return locations

# Get input strings from the user
s = input("Enter the main string to search in: ")
t = input("Enter the second string (the motif to search for): ")

# Find all locations of t in s
locations = FindingMotif(s, t)

# Print the results
if locations:
    print("The motif appears at the following positions:", locations)
else:
    print("The motif was not found in the first string.")
```

```
Enter the main string to search in: CGTAGCGCA
Enter the second string (the motif to search for): G
The motif appears at the following positions: [2, 5, 7]
```

In [9]:

```python
# Task-4 RNA Splicing

def RNASplicing(dna: str, introns: list[str]) -> str:
    """Processes DNA by removing introns and translating to protein.
    Args:
        dna: A DNA sequence string.
        introns: A list of intron sequences to remove.
    Returns:
        The resulting protein string.
    """
    # Remove introns
    for intron in introns:
        dna = dna.replace(intron, '')

    # Convert to RNA
    rna = dna.replace('T', 'U')

    # Use the existing Translation function
    return Translation(rna)

# Get input DNA sequence
dna_sequence = input("Enter the DNA sequence: ")

# Get introns
print("Enter each intron sequence (one per line).")
print("Enter a blank line when done:")

introns = []
while True:
    intron = input()
    if intron.strip() == "":
        break
    introns.append(intron)

# Process and print result
result = RNASplicing(dna_sequence, introns)
print("\nThe resulting protein string is:", result)
```

```
Enter the DNA sequence: ATGGTCTACATAGCTGACAAACAGCACGTAGCAATCGGTCGAATCTCGAGAGGCATATGGTCACATGATCGGTCGAGCGT
GTTTCAAAGTTTGCGCCTAG
Enter each intron sequence (one per line).
Enter a blank line when done:
ATCGGTCGAA
ATCGGTCGAGCGTGT


The resulting protein string is: MVYIADKQHVASREAYGHMFKVCA
```

In [5]:

```python
# Task 5: Finding a Shared Motif

def LongestCommonSubstring(k: list[str]) -> str:
    """Finds the longest common substring of a list of strings.

    Args:
        k: A list of strings.

    Returns:
        The longest common substring.
    """

    # Find the shortest string
    shortest = min(k, key=len)

    # Iterate over all substrings of the shortest string
    # All the codes are pretty straightforward.
    for length in range(len(shortest), 0, -1):
        for start in range(0, len(shortest) - length + 1):
            substring = shortest[start:start+length]
            if all(substring in s for s in k):
                return substring

    return ""

# Get input sequences from the user
sequences_input = input("Enter the sequences (separate by commas): ")
sequences = sequences_input.split(",")

# Find the longest common substring
longest_common_substring = LongestCommonSubstring(sequences)

# Print the result
if longest_common_substring:
    print("The longest common substring is:", longest_common_substring)
else:
    print("There is no common substring among the given sequences.")
```

```
Enter the sequences (separate by commas): AGCTCGCATC, AGCCCTAGC, AGCTTCGAC
The longest common substring is: AGC
```

In [1]:

```python
# Task 6: Finding a Spliced Motif

def FindingSubsequence(s: str, t: str) -> list[int]:
    if not s or not t:
        raise ValueError("Both strings must contain values")

    if len(t) > len(s): #t is shorter than s
        return[]
    position = 0
    output = []
    i=0

    while i < (len(s)):
        if position < len(t) and s[i] ==t[position]:
            output.append(i+1)
            position += 1
            i += 2

        else:
            i +=1
    return output
# Get input
sequence1 = input("Enter sequence 1: ")
sequence2 = input("Enter sequence 2: ")

# Find indices
indices = FindingSubsequence(sequence1, sequence2)

# Print result
print(indices)
```

```
Enter sequence 1: ACGTACGTGACG
Enter sequence 2: GTA
[3, 8, 10]
```

In [15]:

```python
# Task-7: Finding a shared spliced motif
# I used a slightly different approach to this one

def LongestCommonSubsequence(s: str, t: str) -> str:
    m, n = len(s), len(t) # Input lengths

    # Create a 2D table to store LCS lengths
    dp = [[0] * (n + 1) for _ in range(m + 1)] # Creates a 2d table of sizes (m + 1) x (n + 1)

    # Fill the dynamic programming table
    for i in range(1, m + 1): # iterates over the first string
        for j in range(1, n + 1): # iterates over the second string
            if s[i-1] == t[j-1]:
                dp[i][j] = dp[i-1][j-1] + 1 # compares strings and finds matches
            else:
                dp[i][j] = max(dp[i-1][j], dp[i][j-1])

    # Backtrack to find the LCS
    lcs = []
    i, j = m, n
    # Simple loop to backtrace, match and append the matches
    while i > 0 and j > 0:
        if s[i-1] == t[j-1]:
            lcs.append(s[i-1])
            i -= 1
            j -= 1
        elif dp[i-1][j] > dp[i][j-1]:
            i -= 1
        else:
            j -= 1

    # Return the LCS as a string
    return ''.join(reversed(lcs))

def main():
    # Accept sequences directly as input
    print("Enter the first sequence:")
    s = input().strip()

    print("Enter the second sequence:")
    t = input().strip()

    # Find the longest common subsequence
```

```
    lcs = LongestCommonSubsequence(s, t)

    # Print the result
    print("\nLongest Common Subsequence:")
    print(lcs)

if __name__ == "__main__":
    main()
```

```
Enter the first sequence:
AACCTTGG
Enter the second sequence:
ACACTGTGA

Longest Common Subsequence:
ACCTGG
```

In [20]:

```python
#Task-8: Two motifs, one gene:

def ShortestCommonSupersequence(s: str, t: str) -> str:
    lcs = LongestCommonSubsequence(s, t)  # Get the LCS
    scs = []  # To store the shortest common supersequence

    i, j, k = 0, 0, 0  # Pointers for s, t, and lcs
    while i < len(s) or j < len(t):
        # If we have exhausted either string, add the remaining characters
        if k < len(lcs) and i < len(s) and s[i] == lcs[k]:
            scs.append(s[i])
            i += 1
            k += 1
        elif k < len(lcs) and j < len(t) and t[j] == lcs[k]:
            scs.append(t[j])
            j += 1
            k += 1
        elif i < len(s):
            scs.append(s[i])
            i += 1
        elif j < len(t):
            scs.append(t[j])
            j += 1

    return ''.join(scs)

def main():
    # Get user input for the two sequences
    print("Enter the first sequence:")
    s = input().strip()

    print("Enter the second sequence:")
    t = input().strip()

    # Calculate and print the Shortest Common Supersequence
    scs = ShortestCommonSupersequence(s, t)
    print("\nShortest Common Supersequence:")
    print(scs)

if __name__ == "__main__":
```

```
    main()
```

```
Enter the first sequence:
ACGTC
Enter the second sequence:
ATAT

Shortest Common Supersequence:
ACGTCATAT
```