Mumuksha.Pant.HW1

September 23, 2024

1 PART I

1.1 1) Basic Sequence Validation: (10%)

- Write a function called "is_valid_rna" to validate if a given string is a valid RNA sequence (contains only A, C, G, U). The function should return True for valid sequences and False for invalid ones.

Example:For a given sequence "AUGCAUGCAUGC"is_valid_RNA: True

For a given sequence "AUGTXAGCAUGC" is_valid_RNA: False

[192]:
```
import re
def is_valid_rna(rna):
    pattern_to_match= r'^[ACGU]+$'
    result= re.search(pattern_to_match, rna, re.IGNORECASE)
    if result:
        return True

    else:
        return False
```

[193]:
```
#Test cases
print(is_valid_rna("AUGCAUGCAUGC")) # true

print(is_valid_rna("AUGTXAGCAUGC")) #false

print(is_valid_rna("XXYYZ")) #false
print(is_valid_rna("auauau")) #true
```

```
True
False
False
True
```

[ ]:
```
```

[ ]:
```
```

1.2 2)Nucleotide Count: (10%)

- Create a function called 'nucleotide_count' to count the occurrence of each nucleotide (A, C, G, U) in a given RNA sequence.

The function should return a dictionary with nucleotides as keys and their counts as values.

Example:

For a given sequence "AUGCAUGCAUGC"

Nucleotide count: {'A': 3, 'C': 3, 'G': 3, 'U': 3}

[194]:
```python
def nucleotide_count(neocleotide) :
    neocleotide = neocleotide.upper()  # Converting all RNA to upper case
    counts = {'A': 0,  'C': 0, 'G':0 ,'U': 0}
    valid_neocleotides=set(counts.keys())
    for i in neocleotide :
        if i not in valid_neocleotides:
            raise ValueError("Invalid RNA sequence")
        counts[i] += 1
    return counts
```

[195]:
```python
# test cases
try:
    print(nucleotide_count("AUGCAUGCAUGC"))  # valid sequence, upper case
    print(nucleotide_count("auauau"))  # valid sequence ,  lower case
    print(nucleotide_count("XXYY"))  # invalid sequence

except ValueError as e:
    print(e)
```

```
{'A': 3, 'C': 3, 'G': 3, 'U': 3}
{'A': 3, 'C': 0, 'G': 0, 'U': 3}
Invalid RNA sequence
```

[ ]:

[ ]:

[ ]:

1.3 3) Finding Motifs: (10%)

- Write a function called 'find_motifs' to identify and return all occurrences of a given motif (subsequence) within the RNA sequence.

An example of a sequence with Repeated Motifs AUGCUGCAUGCAUGCUGCAUGCAUGCUAG .

The function should handle motifs of varying lengths.

Example: For a given sequence “AUGCAUGCAUGC” Motif ‘AUG’ found at positions: [0, 4,8]

```
[196]: def find_motifs(rna, motif):

           position=[] # list to store the positions of the motifs

           for match in re.finditer(motif, rna, re.IGNORECASE):
               position.append(match.start())  # Add the starting position of each␣
        ↪match

           return position
```

```
[197]: #Test case 1
       find_motifs("AUGCAUGCAUGC", "aug") #valid, lowercase motif
```

```
[197]: [0, 4, 8]
```

```
[198]: #Test case 2
       find_motifs("augcaugCAUGCUGCAUGCAUGCUAG", "AUG") #valid, lowercase RNA
```

```
[198]: [0, 4, 8, 15, 19]
```

```
[199]: #test case 3
       find_motifs("AUGAUGAUGAUG", "AUGAUG") #overlapping sequence
```

```
[199]: [0, 6]
```

```
[200]: #test case 4
       find_motifs("XXYY" , "aug") #motif does not exist in the sequence
```

```
[200]: []
```

1.4 4) Sequence Complementarity: (10%)

- In RNA, A pairs with U and C pairs with G.

Create a function called ‘complementary_sequence’ to generate the complementary sequence of a given RNA sequence by swapping pairs.

Example:

For a given sequence “AUGCAUGCAUGC”

Complementary sequence: UACGUACGUACG

```
[ ]:
```

[220]:
```
def complementary_sequence(rna):
    # pairing rules
    complements = {'A': 'U', 'U': 'A', 'C': 'G', 'G': 'C'}

    # Generate the complementary sequence, swapping pairs
    complementary_seq = ''.join(complements.get(base, base) for base in rna)

    return complementary_seq

#test cases
print(complementary_sequence("AUGCAUGCAUGC"))
print(complementary_sequence("AUGCAUGCAUGC"))   #expected : AUGAUGAUGAUG
print(complementary_sequence("AUGCRY"))   #expected :UACGYR
```

```
UACGUACGUACG
UACGUACGUACG
UACGRY
```

[ ]:
```
```

1.5 5) GC Content Calculation: (10%)

- Write a function called 'gc_content' to calculate the GC content (percentage of nucleotides G and C) in the RNA sequence, which is significant in determining the stability of the molecule. Hint: GC content = GC counts / length of the sequence

Example:For a given sequence "AUGCAUGCAUGC" GC content: 50.0 %

[204]:
```
#This is the normal code I had written previously, without using the regex.
# def gc_content(rna):
#     count_g= {'G':0}
#     count_c= {'C':0}

#     for i in rna:
#         if i=='G':
#             count_g[i]+=1
#         if i=='C':
#             count_c[i]+=1

#     GCcontent= ((count_g['G']+count_c['C'])/len(rna))*100
#     return round(GCcontent, 2)


#Code using regular expression
import re

def gc_content(rna):
    count_g = len(re.findall('G', rna))
    count_c = len(re.findall('C', rna))

    GCcontent = ((count_g + count_c) / len(rna)) * 100
    return round(GCcontent, 2)
```

[205]:
```
#test case
print(gc_content("AUGCAUGCAUGC")) #50.0
print(gc_content("AAUUAAC"))      #14.29
print(gc_content("XXYY"))         #0.0
print(gc_content("CGCGCG"))       #100.0
```

```
50.0
14.29
0.0
100.0
```

[ ]:
```

```

2 PART II

2.1 1) Advanced Sequence Validation: (10%)

- Create a modified version of the 'is_valid_rna' function to also check for commonly used ambiguity codes in RNA sequences (e.g., N for any nucleotide, R for A or G) and validate accordingly.

Example:

valid_sequence_with_ambiguity = "AUGCRYSWKMBDHVN"

invalid_sequence = "AUGTXZGCAUGC"

AGCTU

[206]:
```
import re
def is_valid_rna_modified(rna):
    pattern_to_match= r'^[AGCURYSWKMBDHVN]+$'

    result= re.search(pattern_to_match, rna, re.IGNORECASE)
    if result:
        return True
    else:
        return False
```

[207]:
```
#test case
is_valid_rna_modified("AUGCRYSWKMBDHVN")
```

[207]:
```
True
```

[208]:
```
#test case
is_valid_rna_modified("AUGTXZGCAUGC")
```

[208]:
```
False
```

[209]:
```
#test case
is_valid_rna_modified("AUGCRY")
```

[209]:
```
True
```

[ ]:
```

```

2.2 2) Regex-based Motif Search with Ambiguities: (20%)

- Adapt the 'find_motifs' function to accept motifs with ambiguity codes and identify potential matches in the sequence.

Example:

sequence = "AUGCRYSN"

find_motifs(sequence, "RY")

output: "Motif 'RY' found at positions: [0, 2]

[229]:
```
# Mapping ambiguity codes
ambiguity_codes = {
    'R': '[AG]',     # A or G
    'Y': '[CTU]',    # C, T or U
    'S': '[CG]',     # C or G
    'W': '[ATU]',    # A, T or U
    'K': '[GTU]',    # G, T or U
    'M': '[AC]',     # A or C
    'B': '[CGTU]',   # C, G, T or U
    'D': '[AGTU]',   # A, G, T or U
    'H': '[ACTU]',   # A, C, T or U
    'V': '[ACG]',    # A, C, G
    'N': '[ACGTU]',  # A, C, G, T or U
}

def find_motifs_modified(sequence, motif):

    res = []

    for char in motif:
        if char in ambiguity_codes:
            res.append(ambiguity_codes[char])
        else:
            res.append(char)
    motif_regex = ''.join(res)

    # finding all overlapping match in  sequence
    matches = [match.start() for match in re.finditer(f'(?=({motif_regex}))',␣
↪sequence)]

    return f"Motif '{motif}' found at positions: {matches}"
```

[230]:
```
# Test case 1
sequence = "AUGCRYSN"
motif = "A"
print(find_motifs_modified(sequence, motif))
```

```
Motif 'A' found at positions: [0]
```

[231]:
```
# Test case 2
sequence = "AUGCRYSN"
motif = "RY"
print(find_motifs_modified(sequence, motif))
```

```
Motif 'RY' found at positions: [0, 2]
```

[ ]:
```

```

2.3 3) Sequence Fragmentation and Analysis: (20%)

- Create a function called 'fragment_and_analyze' that fragments the RNA sequence into smaller segments of a specified length and performs a detailed analysis on each fragment including 'is_valid_rna', 'gc_content'and'complementary_sequence'

Example:sequence = "AUGCRYSNAUGCRYXNAUGCRYSN", fragment_length = 6

[215]:
```
def gc_content_modified(rna):
    # Define the GC bases and their contributions for ambiguity codes
    gc_bases = {'G', 'C'}
    ambiguity_codes = {
        'S': 1.0,    # C or G contributes 100%
    }

    gc_count = 0

    for base in rna:
        if base in gc_bases:
            gc_count += 1
        elif base in ambiguity_codes:
            gc_count += ambiguity_codes[base]  # Add partial GC content for␣
↪ambiguity codes
```

```
    # Avoid division by zero
    sequence_length = len(rna)
    if sequence_length == 0:
        return 0.0

    gc_content_percentage = (gc_count / sequence_length) * 100
    return round(gc_content_percentage, 2)
```

[216]:
```
#modifying existing complementary sequence code
def complementary_sequence_modified(rna):
    complementary_sequence_list = []  # result list to store the complementary␣
 ↪sequence

    complement = {
        'A': 'U',
        'U': 'A',
        'C': 'G',
        'G': 'C',
        'S': 'S',  # S = [C or G] so complement remains S
        'N': 'N',  # N can be any base, so complement remains N
        'R': 'Y',  # R = [A or G] so complements to Y
        'Y': 'R'   # Y = [U or C] so complements to R
    }

    return ''.join([complement.get(char, char) for char in rna])

# Test cases
print(complementary_sequence_modified("AUGCAUGCAUGC"))  # Expected: UACGUACGUACG
print(complementary_sequence_modified("AUGCRY"))        # Expected: UACGYR
print(complementary_sequence_modified("GCRYSN"))        # Expected: CGYRNS
```

```
UACGUACGUACG
UACGYR
CGYRSN
```

[217]:
```
def fragment_and_analyze(sequence, fragment_length):
    fragments = [] #to store results

    i = 0
    while i < len(sequence):
        fragment = sequence[i:i + fragment_length]

        #checking valid RNA ( take in account ambiguity codes )
        is_valid = is_valid_rna_modified(fragment)
        #checking gc content only if valid RNA ( take in account ambiguity␣
↪codes )
        gc = gc_content_modified(fragment)  if is_valid else 'N/A'


        #checking complementary sequence only if valid RNA  ( take in account␣
↪ambiguity codes )
        comp_seq = complementary_sequence_modified(fragment) if is_valid else␣
↪'N/A'

        #add all the results to fragments.
        fragments.append({
            'fragment': fragment,
            'is_valid_rna': is_valid,
            'gc_content': gc,
            'complementary_sequence': comp_seq
        })

        i += fragment_length #move to next

    return fragments
```

[218]:
```
#test case
fragment_and_analyze("AUGCRYSNAUGCRYXNAUGCRYSN", 6)
```

[218]:
```
[{'fragment': 'AUGCRY',
  'is_valid_rna': True,
  'gc_content': 33.33,
  'complementary_sequence': 'UACGYR'},
 {'fragment': 'SNAUGC',
  'is_valid_rna': True,
  'gc_content': 50.0,
  'complementary_sequence': 'SNUACG'},
 {'fragment': 'RYXNAU',
  'is_valid_rna': False,
  'gc_content': 'N/A',
  'complementary_sequence': 'N/A'},
 {'fragment': 'GCRYSN',
  'is_valid_rna': True,
  'gc_content': 50.0,
  'complementary_sequence': 'CGYRSN'}]
```

[219]:
```
#test case
fragment_and_analyze("AUGCRY",2)
```

[219]:
```
[{'fragment': 'AU',
  'is_valid_rna': True,
  'gc_content': 0.0,
  'complementary_sequence': 'UA'},
 {'fragment': 'GC',
  'is_valid_rna': True,
  'gc_content': 100.0,
  'complementary_sequence': 'CG'},
 {'fragment': 'RY',
  'is_valid_rna': True,
  'gc_content': 0.0,
  'complementary_sequence': 'YR'}]
```

[ ]: