Juliana Camila Rincón Rojas 20231020013

Análisis de sistemas

13/03/2024

```
[ ]  import random

     def create_sequence():
       nucleotid_bases = ['A','C','G','T']
       size_sequence = random.randint(10,20)
       new_sequence = [nucleotid_bases[random.randint(0,3)] for i in range(size_sequence)]
       return "".join(new_sequence)

     print(create_sequence())
```

```
GCTCTCTCCGAAGGCCCC
```

```
[ ]  def create_database():
       db_size = 50000
       data_base = [create_sequence() for i in range(db_size)]
       return data_base

     print(create_database())
```

In the first function, defined as create_sequence, the nucleotide bases are defined and a new sequence is created with these bases randomized. In the second function, a database is created filled with the random sequences generated earlier.

```
def get_combinations(n, sequence, bases):
  if n == 1:
    return [sequence+bases[i] for sequence in sequence for i in range(len(bases))]
  else:
    sequence_ = [sequence+bases[i] for sequence in sequence for i in range(len(bases))]
    return get_combinations(n-1, sequence_, bases)

def count_motif(motif, sequences_db):
  count = 0
  for sequence in sequences_db:
    count += sequence.count(motif)
  return count

def get_motif(motif_size, sequences_db):
    nucleotid_bases = ['A','C','G','T']
    combinations = get_combinations(motif_size, [""], nucleotid_bases)
    max_counter = 0
    motif_winner = ""
    for motif_candidate in combinations:
      temp_counter = count_motif(motif_candidate,sequences_db)
      if temp_counter > max_counter:
        max_counter = temp_counter
        motif_winner = motif_candidate
    return motif_winner

motif_Six = get_motif(6, create_database())
motif_Eight = get_motif(8, create_database())
```

The `get_combinations` function generates all possible sequences of length `n` using the given `bases` and starting with the initial `sequence`.

The `count_motif` function counts the occurrences of a given `motif` in a list of `sequences_db` sequences.

The `get_motif` function finds the motif of a specific size (`motif_size`) that appears most frequently in a given list of `sequences_db`.

It returns the motif with the highest count found for each specified motif size (`6` and `8` in this case) in the `sequences_db` created by `create_database()`.

```
import math

def shannon_entropy_six(seq):
  counts  = {base: sequence.count(base) for base in ['A','G','C','T']}
  total_bases = sum(counts.values())
  entropy = 0
  for count in counts.values():
    if count > 0:
      prob = count / total_bases
      entropy -= prob * math.prob(log2())
  if entropy > 0.9 || entropy < 2:
    return entropy
  else:
    return shannon_entropy(get_motif(6, create_database()))

  def shannon_entropy_eight(seq):
  counts  = {base: sequence.count(base) for base in ['A','G','C','T']}
  total_bases = sum(counts.values())
  entropy = 0
  for count in counts.values():
    if count > 0:
      prob = count / total_bases
      entropy -= prob * math.prob(log2())
  if entropy > 0.9 || entropy < 2:
    return entropy
  else:
    return shannon_entropy(get_motif(8, create_database()))

print(shannon_entropy_six(motif_Six))
print(shannon_entropy_eight(motif_eight))
```

These functions calculate the Shannon entropy for DNA sequences of length six and eight, respectively. They first count the occurrences of each base in the sequence, then use this information to calculate the Shannon entropy. If the calculated entropy is between 0.9 and 2, it is returned; otherwise, the entropy is recursively calculated using `get_motif(6, create_database())` or `get_motif(8, create_database())` until a valid entropy value is obtained. Finally, the script prints the calculated Shannon entropy for `motif_Six` and `motif_Eight`.

The range of entropy between 0.9 and 2 is chosen because very low values could indicate lack of genetic diversity, while very high values could suggest measurement errors or random sequences. This range is considered suitable for indicating sufficient but not extreme genetic diversity in the analyzed DNA sequences.

https://colab.research.google.com/drive/12SwJZQgH_JjG0nUA8s54JK3gHyiPJ_au?usp=sharing