Universidad de Aysén

Desafío III

Empresa Biocorp

Nombre: Karla Legue

Profesor: Gabriel Núñez

Fecha: 19/12/2022

Descripción

El desafío asignado consistía en generar un código en lenguaje Python, el cual a partir de la lectura de un archivo fasta que contiene un respectivo código genético, traducía este a una secuencia de proteína. Luego, la empresa requería de estadísticas tales como, el número de nucleótidos, contabilización de codones y cantidad de aminoácidos presentes. Por último, también se solicitaba un gráfico que mostrara las relaciones entre codones y aminoácidos.

Requerimiento 1

El primer requerimiento consistía en poder realizar la lectura correcta del archivo, sin considerar el título incluido, cumplir con la traducción de manera acertada, es decir no considerar la secuencia no codificante del código genético, reemplazar los nucleótidos timina por uracilo, y considerar los codones start y stop para realizar dicha traducción.

Antes de comenzar, se generó un diccionario. En donde, las keys eran los codones y el valor eran los aminoácidos correspondientes a este.

```
tabla_codones={
    "GCU": "A", "GCC": "A", "GCA": "A", "GCG": "A", "CGU": "R",
    "CGC": "R", "CGA": "R", "CGG": "R", "AGA": "R", "AGG": "R",
    "AAU": "N", "AAC": "N", "GAU": "D", "GAC": "D", "UGU": "C",
    "UGC": "C", "CAA": "Q", "CAG": "Q", "GAA": "E", "GAG": "E",
    "GGU": "G", "GGC": "G", "GGA": "G", "GGG": "G", "CAU": "H",
    "CAC": "H", "AUU": "I", "AUC": "I", "AUA": "I", "AUG": "START",
    "UUA": "L", "UUG": "L", "CUU": "L", "CUC": "L", "CUA": "L",
    "CUG": "L", "AAA": "K", "AAG": "K", "AUG": "M", "UUU": "F",
    "UUC": "F", "CCU": "P", "CCC": "P", "CCA": "P", "CCG": "P",
    "UCU": "S", "UCC": "S", "UCA": "S", "UCG": "S", "AGU": "S",
    "AGC": "S", "ACU": "T", "ACC": "T", "ACA": "T", "ACG": "T",
    "UGG": "W", "UAU": "Y", "UAC": "Y", "GUU": "V", "GUC": "V",
    "GUA": "V", "GUG": "V", "UAG": "STOP", "UGA": "STOP", "UAA": "STOP"}
```

- Para leer el archivo.

```
archivo =open(input ("Ingrese el nombre de su archivo: "))
nombre = archivo.readline()##Aquí un salto para evitar leer nombre del archivo##
lineas =archivo.read()##Leer archivo##
```

En las líneas anteriores, el usuario digita el nombre del archivo y con la función open() éste se abre. Luego, con la función readline() se genera un salto de carro para no considerar en la lectura el título del archivo.

Después en la variable “líneas” se guarda la lectura del archivo como una cadena de texto (string**)**

- Para cambiar timina por uracilo

```
lineas= lineas.replace ("T", "U")##Reemplazo timina por uracilo
```

Para poder reemplazar dichos aminoácidos, se utilizó la función replace (), esta función se puede utilizar de manera correcta porque líneas es un string.

- No considerar secuencia no codificante (intrón, I-F)

```
lineaslimpia = ""
```

```
i = 0
while i < len(lineas):
    if lineas[i]=='I':
        while lineas[i]!='F':
            i = i + 1
    else:
        lineaslimpia = lineaslimpia + lineas[i]
    i= i + 1
print (lineaslimpia)
```

Se inicializa una cadena de texto vacía llamada “lineaslimpia”, la cual se encargará de almacenar los codones que no estén considerados entre la secuencia no codificante, incluyendo el I y el F.

Para ello, inicializa una variable i en 0, para después realizar un ciclo while el cual establecerá como condición que mientras i sea menor al largo total de líneas (cadena de texto). Dentro de este bucle preguntará si líneas[i] es decir, si en la posición líneas [i] es igual a “I” entonces entrará al otro ciclo anidado buscando hasta encontrar la “F”, si no es encontrada el ciclo irá aumentando de 1 en 1. De forma contraria, si no es encontrado el I, la letra correspondiente se irá almacenando en líneas limpias.

- Búsqueda codón AUG - codón START

```
codones = ""
i = 0
while i< len(lineaslimpia)-2:##Busqueda codón AUG##
    if codon_inicio [0]==lineaslimpia[i] and codon_inicio [1] == lineaslimpia[i+1] and codon_inicio [2]== linea
        while i< len(lineaslimpia)-2:
            codones = codones + lineaslimpia[i:i+3]
            i = i + 3
    i = i + 1
print(codones)
```

Se inicializará codones como un string vacío, inicializando la variable i en 0. Luego, en el ciclo se establece la condición de mientras i sea menor al largo de líneas limpias-2, entrará al ciclo en donde comparará el codón inicio con líneas limpias hasta encontrar las 3 primeras letras “A, “U”, G”.

- Generar secuencia de proteína y representarlo en un archivo de salida

```
sequence_proteina = ""
for i in range (0, len(codones),3):
    codon = codones[i]+codones[i+1]+codones[i+2]
    sequence_proteina= sequence_proteina+tabla_codones[codon]
print (sequence_proteina)
archivo_creado = open("SEQUENCE_1_PROTEINA", "w")
archivo_creado.write (sequence_proteina)
archivo_creado.close()
```

Con un ciclo for se representa desde la posición 0 hasta el largo de codones, con salto de 3. Luego, en la variable codon se almacena, las letras de cada codón. Pata así en sequence_proteina se concatena lo que se tiene en sequence_proteina más el valor de la key codones.

Después, para poder generar el archivo de salida, se abre un archivo con el nombre sequence_1_proteina, donde su contenido será lo almacenado en sequence_proteina.

Requerimiento 2

El segundo requerimiento consistía en generar estadísticas en base a las secuencias válidas que se generaron.

1. **Número y porcentaje de nucleótidos (A,C,U,G) por secuencia valida.**

```
total_nucleotidos= 0
nucleotido_A= 0
nucleotido_C = 0
nucleotido_U = 0
nucleotido_G = 0
for i in sequence_proteina:
    if i== "A":
        nucleotido_A= nucleotido_A+1
    elif i== "C":
        nucleotido_C= nucleotido_C+1
    elif i== "U":
        nucleotido_U = nucleotido_U+1
    elif i== "G":
        nucleotido_G = nucleotido_G+1

    total_nucleotidos = (nucleotido_A+nucleotido_C+nucleotido_U+nucleotido_G)
    porcentaje_A= (nucleotido_A*100)/total_nucleotidos
    porcentaje_C = (nucleotido_C*100)/total_nucleotidos
    porcentaje_U = (nucleotido_U*100)/total_nucleotidos
    porcentaje_G = (nucleotido_G*100)/total_nucleotidos
print ("La cantidad de nucleotidos A es de", (nucleotido_A-2), "y su porcentaje es de",round(porcentaje_A), "%
```

Para contar el número de nucleótidos se inician contadores en cero.

```
nucleotido_A= 0
nucleotido_C = 0
nucleotido_U = 0
nucleotido_G = 0
```

Después, se inicia un ciclo for en donde i será el elemento que vaya recorriendo lo almacenado en sequence_proteina. En donde si i ==A entonces el contador nucleótido_A, irá aumentando en 1 y así con el resto de nucleótidos. Luego, para sacar el porcentaje de ellos se realiza un total, que almacena y suma lo contenido en los 4 nucleótidos para luego realizar la operación de porcentaje con ello.

```
nucleotido_A= 0
nucleotido_C = 0
nucleotido_U = 0
nucleotido_G = 0
for i in sequence_proteina:
    if i== "A":
        nucleotido_A= nucleotido_A+1
    elif i== "C":
        nucleotido_C= nucleotido_C+1
    elif i== "U":
        nucleotido_U = nucleotido_U+1
    elif i== "G":
        nucleotido_G = nucleotido_G+1

    total_nucleotidos = (nucleotido_A+nucleotido_C+nucleotido_U+nucleotido_G)
    porcentaje_A= (nucleotido_A*100)/total_nucleotidos
    porcentaje_C = (nucleotido_C*100)/total_nucleotidos
    porcentaje_U = (nucleotido_U*100)/total_nucleotidos
    porcentaje_G = (nucleotido_G*100)/total_nucleotidos
print ("La cantidad de nucleotidos A es de", (nucleotido_A-2), "y su porcentaje es de",round(porcentaje_A
```

2. Número y porcentaje de codones encontrados

```
    total_nucleotidos = (nucleotido_A+nucleotido_C+nucleotido_U+nucleotido_G)
    porcentaje_A= (nucleotido_A*100)/total_nucleotidos
    porcentaje_C = (nucleotido_C*100)/total_nucleotidos
    porcentaje_U = (nucleotido_U*100)/total_nucleotidos
    porcentaje_G = (nucleotido_G*100)/total_nucleotidos
print ("La cantidad de nucleotidos A es de", (nucleotido_A-2), "y su porcentaje es de",round(porcentaje_A), "%
###Numero y porcentaje de codones por secuencia válida
```

3. Número de aminoácido

```
##NUMERO DE AMINOACIDOS##
listapolares_sin_carga= ["S", "T", "C", "Y", "N", "Q"]
listapolares_con_cargapositiva= ["H", "R", "K"]
listapolares_con_carganegativa= ["D", "E"]
lista_apolares = ["G", "A", "V", "L", "I", "F", "W", "M", "P"]
polares_sin_carga = 0
polares_con_carga_positiva= 0
polares_con_carga_negativa = 0
apolares = 0
i = 0
for i in sequence_proteina:
    if i in listapolares_sin_carga:
        polares_sin_carga = polares_sin_carga +1
    if i in listapolares_con_cargapositiva:
        polares_con_carga_positiva = polares_con_carga_positiva +1
    if i in listapolares_con_carganegativa:
        polares_con_carga_negativa = polares_con_carga_negativa +1
    if i in lista_apolares:
        apolares = apolares+1
print ("La cantidad de polares sin carga es de", polares_sin_carga,"\nLa cantidad de polares con carga positiva
```

Para poder contabilizar los aminoácidos, generé 4 listas, en donde se encontraban los elementos correspondientes a polares sin carga, polares con carga positiva, negativa y apolares. Luego, se inician contadores con los nombres requeridos a cada aminoacido.

Luego, con un ciclo for la variable i va recorriendo sequence_proteina, en donde si encuentra una de las letras correspondientes, lo va a agregando a polares_sin_carga, polares_con_carga_negativa, polares_con_carga_positiva o apolares según corresponda.

Requerimiento 3

```
    g = Digraph( 'G' , filename= grafico.gv ',format= png ')
    g.attr('node', shape='circle')


    i = 0
    while i+6 !=len(codones):
        g.attr('node', shape='circle')
        g.edge(codones[i:i+3],codones[i+3:i+6])
        i += 3

    g.attr('node', shape='circle')
    g.edge(codones[i:i+3],codones[i+3:i+6])


    i = 0
    while i+3 !=len(codones):
        g.attr('node', shape=('rectangle'))
        g.edge(codones[i:i+3], str(dic.get(codones[i:i+3])), color='red')
        i += 3



    g.view()
print ("Ha cerrado la sesión")
```