GUIA DE LABORATORIO PARA BIOINFORMÁTICA

GUIA 1: INTRODUCCIÓN A PYTHON PARA BIOINFORMÁTICA

OBJETIVOS DE LA PRÁCTICA

El estudianteal finalizarestapráctica de laboratorio, será capaz de:

- Entender y utilizar python y tipos de datos diversos para solucionar problemas bioinformáticos.
- Escribir programas que efectúencálculos para resolver problemas biológicos.

TRABAJO PREPARATORIO

Para un trabajo con resultados satisfactorios, es necesario que el estudiante:

- Tenga familiaridad con el lenguaje deprogramación Python.

MATERIALES NECESARIOS

Para el desarrollo dela presente guía es necesario:

- Tener Python instalado o en su defecto usar un intérprete en línea.

MARCO TEÓRICO

Python se utiliza para varias tareas en bioinformática, incluida la investigación académica, la manipulación de datos, la secuenciación de proteínas, el análisis de datos, la visualización de datos, el acceso a bases de datos y el aprendizaje estadístico. También se utiliza para análisis deestructuras macromoleculares, análisis desecuencias de ADN y análisis de datos de microarrays.

La mayoría de los especialistas en bioinformática o biólogos no saben programar y prefieren dedicar su tiempo a otras tareas. Esto hace que Python sea ideal para ellos debido a las numerosas bibliotecas disponibles que agilizan el proceso de programación. Estos profesionales también loencuentran útil para crear prototiposmás rápidos, depurarmásfácilmente y obtener resultados másfructíferos.

La vida es definitivamente digital. El código genético de todos los organismos vivos está representado por una larga secuencia de moléculas simples llamadas nucleótidos o bases, que conforman el ácido desoxirribonucleico, más conocido como ADN. Sólo hay cuatro nucleótidos de este tipo, y el código genético completo de un ser humano puede verse como una cadena simple, aunque de 3 mil millones de longitud, de letras A, C, G y T

Analizar los datos del ADN para obtener una mayor comprensión biológica es mucho sobre la búsqueda en cadenas (largas) de ciertos patrones decadenas que involucran las letras A, C, G y T. Esto es una parte integral de la bioinformática, una disciplina científica que abordael uso de computadoras para buscar, explorar y usar información sobre genes, ácidos nucleicos y proteínas.

LIBRERÍAS CLAVE PARA BIOINFORMÁTICA

Python tiene amplias bibliotecas que le resultan útiles tanto como principiante como como experto en el campo. Para cualquiertarea que desee realizar utilizando Python, es casi seguro que encontrará una biblioteca para ello. A continuación, se enumeran algunas bibliotecas de Python para bioinformática.

BioPython. BioPython es una herramienta decódigo abierto en Python creada por una coalición internacional dedesarrolladores. Es una recopilaciónde herramientas python utilizadas para biología computacional y bioinformática.

Esta es una biblioteca ideal para usar al aprender Python para bioinformática y documentaciónde características, alineaciónde secuencias y código fuente.

PyCogent. PyCogent es una biblioteca de software para biología genómica que se utiliza para dar sentido a secuencias biológicas y genómicas.

Biskit. Biskit es otrabiblioteca de python decódigo abierto que se puede utilizar para la investigación, manipulación y análisis de bioinformática estructural de estructuras macromoleculares, complejos deproteínas y trayectorias dedinámica molecular.

Galaxy. Galaxy es una plataforma abierta basada en web para investigaciones biomédicas con uso intensivo de datos con una comunidad de expertos a quienes les gusta compartir sus hallazgos. Es fácil de usar, reproducible y transparente.

pyMOL. PYMOL es un software de visualización molecular de código abierto que se puede utilizar para renderizar y animarestructuras moleculares en 3D.

NOCIONES DE ADN

El ADN constade una serie de nucleótidosA, T, C, G.

A y T son complementariosentresí, y C y G son complementariosentresí. Entonces, una hebra de ADN que consta devarios alfabetos, como "AGCAATGC", se uniría a otra hebra formada por sus complementos: "TCGTTACG".

Cuando ocurre la replicación, estas 2 hebras se separan y crean su propio par complementario. Así obtenemosahora 2 pares de ADN similares.

Dado que el ADN en sí es una colecciónde grandes cantidades de datos, comprender el ADN se convierte en un problema interesante de resolver y utiliza matemáticas y programación.

5. EJERCICIOS RESUELTOS

A. Escribir un programa que cuenteel número de apariciones de un determinadopatrón en un textodeterminado. Por ejemplo, en "CACCGTCACAC", el patrón "CAC" aparece 3 veces.

```
def contar_ocurrencias1(texto, patron):
    return len(list(filter(lambda i: texto[i:i + len(patron)] == patron,
range(len(texto) - len(patron) + 1))))

def contar_ocurrencias2(texto, patron):
    return sum(1 for i in range(len(texto) - len(patron) + 1) if
texto[i:i+len(patron)] == patron)
```

Prueba el correcto funcionamiento de los dos métodos. ¿Cuál de los dos te parece mejor y por qué?

```
[2]
def contar_ocurrencias1(texto,patron):
  return len(list(filter(lambda i:texto[i:i +len(patron)]==patron,range(len(texto)-len(patron)+1))))
def contar_ocurrencias2(texto,patron):
  return sum(1 for i in range(len(texto)-len(patron)+1) if texto[i:i+len(patron)]==patron)

Texto = "CACCGTCACAC"
patron = "CAC"


print(contar_ocurrencias1(Texto,patron))
print(contar_ocurrencias2(Texto,patron))
```

```
3
3
```

El segundo es el mas adecuado en cuanto a eficiencia porque directamente alla la suma si Encuentra una coincidencia, no como el primero que filtra todavia los resultados los convierte en Una lista y recien halla los valores que se repiten.

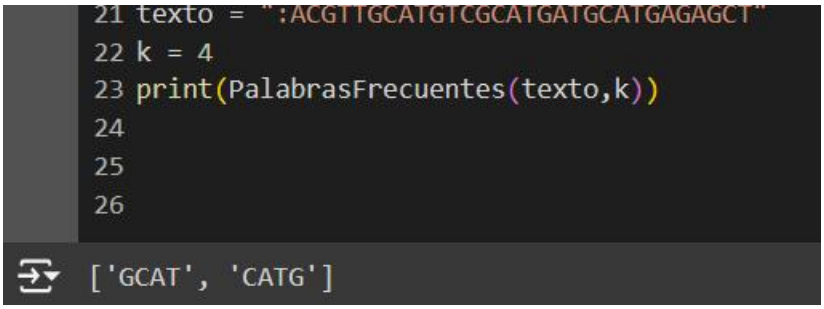
B. Una subcadena de longitud k en un texto determinado se denomina k-mer del texto. Dado un texto y un número k, en este problema, tuvimos que filtrar las palabras (o kmers o subcadenas) que aparecen con másfrecuencia de una longitud k determinada. A continuación se muestran ejemplos de entrada y salida.

```
Entrada
Texto: ACGTTGCATGTCGCATGATGCATGAGAGCT
Longitud: 4
Salida
CATG GCAT
```

```
def MapaFrecuencia(Texto, k):
    frecuencia = {}
    n = len(Texto)
    for i in range(n-k+1):
        Patron = Texto[i:i+k]
        frecuencia[Patron] = 0
    for i in range(n-k+1):
        Patron = Texto[i:i+k]
        frecuencia[Patron] += 1
    return frecuencia
```

```python
def PalabrasFrecuentes(Texto, k):
    palabras = []
    frecuencia = MapaFrecuencia(Texto, k)
    max_frecuencia = max(frecuencia.values())
    for clave in frecuencia:
        if frecuencia[clave] == max_frecuencia:
            palabras.append(clave)
    return palabras
```

Explica en un párrafo cómo funciona el algoritmo y prueba su correcto funcionamiento.



El modulo de mapafrecuencia se encarga de contar cuantas veces aparece cada subsecuencia de longitud k en un texto dado.

El modulo de palabrasFrecuentes encuentra las subsecuencias que mas se repiten en un texto.

En conjunto, estos modulos son utiles para encontrar patrones repetitivos en secuencia de texto.

C. Los nucleótidos A y T son complementariosentresí, al igual que C y G. Por lotanto, un nucleótido A estaría emparejado con T y G estaría emparejado con C. Si tenemos una hebra, digamos AG, ¿cuál sería su complemento? La respuesta es un poco sorprendente.

No es TC como cabría esperar. De hecho es CT. Esto se debe a que la hebra y su complemento par corren en direcciónopuesta. Para encontrarel complementode una hebra, necesitamos encontrar el complemento de cada letra y luego invertir la dirección.

AG → TC→ CT

GACGTAT → CTGCATA → ATACGTC

```python
def ComplementoInverso(Patron):
    return invertir(complemento_dna(Patron))

def complemento_dna(patron):
    return ''.join(map(complemento_caracter_dna, patron))

def invertir(s):
    return s if len(s) == 0 else invertir(s[1:]) + s[0]

def complemento_caracter_dna(caracter):
    return {"A": "T", "T": "A", "G": "C", "C": "G"}[caracter]
```

Explica en un párrafocómofunciona elalgoritmo y prueba su correctofuncionamiento.

El algoritmo calcula el complemento inverso de una secuencia de ADN. Usando:

1. Como modulo el **ComplementoInverso(Patron):** Esta es la funcion principal que combina dos pasos:

-Calcula el complemento de la secuencia de ADN

2. **complemento_dna(patron)**:

Esta función convierte cada nucleótido de la secuencia en su nucleótido complementario, de acuerdo a las reglas del ADN.

3. **invertir(s)**:

Esta función invierte la cadena de ADN de manera recursiva.

4. Función **complemento_caracter_dna**:

la función complemento_caracter_dna toma un nucleótido (representado por el carácter) y devuelve su complementario de acuerdo con las reglas definidas.

En resumen Los cuatro módulos (funciones) trabajan en conjunto para calcular el complemento inverso de una secuencia de ADN.

```
[7]  1 def ComplementoInverso(Patron):
     2   return invertir(complemento_dna(Patron))
     3 def complemento_dna(patron):
     4   return''.join(map(complemento_caracter_dna,patron))
     5 def invertir(s):
     6   return s if len(s)==0 else invertir(s[1:])+s[0]
     7 def complemento_caracter_dna(caracter):
     8   return {"A":"T","T":"A","G":"C","C":"G"}[caracter]
     9
    10
    11 patron = "ATGC"
    12 complemento_inverso = ComplementoInverso(patron)
    13 print(complemento_inverso)

       GCAT
```

6. EJERCICIOS PROPUESTOS

A. Escribir un programa en Python que divida una cadena de ADN (string) en subcadenas de tamaño fijo.

B. Escribir un programa que una n cadenas de ADN en una sola cadena y reemplazar un nucleótido A por un T en la cadena resultante.

C. Hacer un módulo que transcriba una cadena de ADN a ARN.

Una cadena de ARN es una cadena formada por el alfabeto que contiene «A», «C», «G» y «U».

Dada una cadena de ADN *t* correspondiente a una cadena codificante, su cadena de ARN transcrita *u* se forma sustituyendo todas las apariciones de «T» en t por «U» en *u*

D. Investiga y resume en un párrafo que es y para qué sirvela biblioteca de Python "Biopython".

Biopython es una biblioteca de Python diseñada para facilitar el trabajo con datos biológicos y bioinformáticos. Proporciona herramientas para la manipulación de secuencias de ADN, ARN y proteínas, además de funcionalidades para realizar análisis complejos como alineamientos de secuencias, búsqueda en bases de datos biológicas, análisis filogenético y manipulación de estructuras de proteínas. Es especialmente útil en proyectos de biología computacional, ya que simplifica la interacción con formatos de archivo comunes en bioinformática (como FASTA y GenBank), y permite a los investigadores trabajar con datos biológicos de manera eficiente en entornos de programación.