E1.Actividad Integradora

Víctor Misael Escalante Alvarado, A01741176
Rodrigo López Murguía, A01742780

Resumen -.

I. INTRODUCCIÓN

Este documento presenta la solución a la primera entrega pertinente a la situación problema de la materia Análisis y diseño de algoritmos avanzados. Presentando una descripción de la problemática a resolver, en la cual se aborda la importancia e impacto que presenta esta problemática, se termina llevando a cabo un reporte sobre las técnicas de programación utilizadas para llegar a la solución satisfactoria para esta entrega.

II. DESCRIPCIÓN DEL PROBLEMA

El objetivo principal de esta actividad es realizar una comparación entre 2 genomas del virus SARS-CoV-2 obtenidos de estudios de Wuhan en 2019 y Texas 2020. Se busca evaluar cambios significativos en la estructura de los genomas, estos cambios en el virus podrían estar asociados con cambios en las características del virus, como la transmisibilidad, la respuesta a tratamientos, entre otras. Este análisis ayudará a obtener información clave para la comprensión de la evolución del virus y su impacto en la salud de la sociedad.

III. METODOLOGÍA

Para la resolución de este reto se ha utilizado una aproximación perteneciente al paradigma de clases y funciones. Esto dado a que , por el contexto del problema, las distintas soluciones se componen de reutilizar ciertos aspectos en común , mismos aspectos que pueden ser contenidos en una clase y, partiendo de las características contenidas , podemos trabajar con ellas por medio de funciones.

A. Clases Utilizadas

Se ha decidido implementar una clase Trie que representa el árbol trie que nace de la lectura de los archivos que contienen las cadenas de caracteres representando las cadenas de ARN de los virus. A medida que se van registrando los caracteres dentro de la cadena estos son guardados en una clase nodo dentro de la cual se encuentran los datos necesarios para trabajar las distintas funciones que resuelven los puntos mencionados en el apartado de **Resultados.**

1. Clase Trie

```
class Trie:
    def __init__(self):
        self.raiz = Nodo("#")

    def agregar_palabra(self, palabra, aminoacid):
        actual = self.raiz
        for letra in palabra:
            hijo = actual.search_hijo(letra)
            if hijo is None:
                hijo = actual.agregar_hijo(letra)
            actual = hijo
        actual.isFinal = True
        actual.aminoacid = aminoacid

    def mostrar_arbol(self):
        actual = self.raiz
        queue = deque([actual])

        while queue:
            for _ in range(len(queue)):
                nodo = queue.popleft()
                print(nodo, end=" ")
                for hijo in nodo.hijos:
                    queue.append(hijo)
            print()
```

```
def buscar_palabra(self, palabra):
    actual = self.raiz
    for letra in palabra:
        hijo = actual.search_hijo(letra)
        if hijo is None:
            return None
        else:
            actual = hijo
    if actual.isFinal:
        return actual.aminoacid
    else:
        #Imprime las palabras posibles
        print("Autocompletado para " + palabra + ":")
        palabras = buscar_palabras(actual, palabra)
        for palabra in palabras:
            if palabra == palabras[-1]:
                print(palabra)
            else:
                print(palabra, end=", ")
        return palabras
```

1. [Definición de la clase trie]

2. Clase nodo

```
class Trie:
    def __init__(self):
        self.raiz = Nodo("#")

    def agregar_palabra(self, palabra, aminoacid):
        actual = self.raiz
        for letra in palabra:
            hijo = actual.search_hijo(letra)
            if hijo is None:
                hijo = actual.agregar_hijo(letra)
            actual = hijo
        actual.isFinal = True
        actual.aminoacid = aminoacid

    def mostrar_arbol(self):
        actual = self.raiz
        queue = deque([actual])

        while queue:
            for _ in range(len(queue)):
                nodo = queue.popleft()
                print(nodo, end=" ")
                for hijo in nodo.hijos:
                    queue.append(hijo)
            print()
```

```
def buscar_palabra(self, palabra):
    actual = self.raiz
    for letra in palabra:
        hijo = actual.search_hijo(letra)
        if hijo is None:
            return None
        else:
            actual = hijo
    if actual.isFinal:
        return actual.aminoacid
    else:
        #Imprime las palabras posibles
        print("Autocompletado para " + palabra + ":")
        palabras = buscar_palabras(actual, palabra)
        for palabra in palabras:
            if palabra == palabras[-1]:
                print(palabra)
            else:
                print(palabra, end=", ")
        return palabras
```

2. [Definición de la clase nodo]

Debido a las características particulares de cada proteína se tomó la decisión de asignar una clase en la cual contenemos su nombre y secuencia.

```
class Protein:
    def __init__(self, name, sequence):
        self.name = name
        self.sequence = sequence

    def __str__(self):
        return f"{self.name}: {self.sequence}"
```

3. [Definición de la clase Protein]

B. Funciones Diseñadas

En base a las características de la clase trie se pueden diseñar las funciones

[StringMatchZ O(n)]

Función diseñada para obtener los índices de aparición de una subcadena determinada dentro de otra.Para llevar a cabo esto se hace uso de un zarray (definido por la función zarray explicada a continuación) con el cual se enlistan los índices donde coincide el valor con la longitud de la subcadena.

```
def StringMatchZ(texto, patron):
  m = len(texto)
  n = len(patron)
  m = m + n + 1
  indexes = []
  zArray = ZArray(texto,patron)
  for i in range(m):
    if zArray[i] == n:
      # return i - n - 1
      indexes.append(i-n-1)
  return indexes
```

4.[Definición de la función StringMatchZ]

[ZArray O(n+m)]

Función diseñada para generar el arreglo z de una subcadena dentro de una cadena.El arreglo z es definido por el número de coincidencias de una subcadena dentro de otra cadena. Funciona por medio de identificar las coincidencias y de contabilizar estas mismas.

```
#Z array para string-matching
def ZArray(texto, patron):
    # Z - Array Algorithm
    n = len(texto)
    m = len(patron)

    L = 0
    R = 0
    x = n + m + 1
    Z = [-1 for i in range(x)]
    C = patron + "$" + texto
```

```
for i in range(x):
    if i <= R:
        k = i - L
        if Z[k] < R - i + 1:
            Z[i] = Z[k]
        else:
            L = i
            while R < x and C[R-L] == C[R]:
                R += 1
            Z[i] = R - L
            R -= 1
    else:
        L = R = i
        while R < x and C[R-L] == C[R]:
            R += 1
        Z[i] = R - L
        R -= 1
return Z
```

5.[Definición de la función ZArray]

[Manacher O(n*n)]

Función diseñada para buscar el palíndromo más largo dado una cadena de caracteres , en este caso siendo la cadena el gen determinado de un virus. Para llevar a cabo la búsqueda del pal´ndromo se recorre el arreglo de manera lineal mientras se encuentra con un carácter pivote al cual se le comparan sus vecinos hasta acumular el máximo palíndromo posible

```
#Manacher para palíndromo más largo en los genes
def Manacher(frase):
    centro = limite = 0

    texto = "@$" + "$".join(frase) + "$#"
    e = len(texto)

    P = [0] * e

    for i in range(1, e - 1):
        if i < limite:
            simetrica = 2 * centro - i
            P[i] = min(limite - i, P[simetrica])

        while (i + P[i] + 1 < e and
               i - P[i] - 1 >= 0 and
               texto[i + P[i] + 1] == texto[i - P[i] - 1]):
            P[i] += 1
```

```
        if i + P[i] > limite:
            limite = i + P[i]
            centro = i

    maxVal = max(P)
    maxIndex = P.index(maxVal)
    inicio = (maxIndex - maxVal) // 2
    fin = inicio + maxVal
    return frase[inicio:fin], inicio, fin
```

[Search Proteinas O(n)]

Función diseñada para generar una lista de las proteínas pertenecientes a una cadena, en este caso se recorre por completo el arreglo de aminoácidos y se genera una lista de los codones conforme se va recorriendo el arreglo

```
# Buscar proteinas en secuencia
def SearchProteinas(sequence):
  lista_aminoacidos = []
  reading_frames = [0,1,2]

  # Por cada inicio de proteina, traducir hasta encontrar un STOP o hasta que acabe
  while len(reading_frames) > 0:
      start_index = reading_frames[0]
      end_index = reading_frames[0]

      protein = ""
      for i in range(start_index, len(sequence), 3): # Itera de 3 en 3 para buscar los codones
        codon = sequence[i:i+3]
        if len(codon) == 3:
          aminoacid = diccionario_aminoacidos.buscar_palabra(codon)
          protein += aminoacid
          end_index = i+3

      lista_aminoacidos.append((reading_frames[0], protein))
      reading_frames.pop(0)

  results = []

  # Buscar las proteinas en las listas encontradas
  # for aminoacid in lista_aminoacidos:
  #    for protein in proteins:
  for protein in proteins:
    encontrado = False
    for aminoacid in lista_aminoacidos:
      z = StringMatchZ(aminoacid[1], protein.sequence)
      if len(z) > 0:
        encontrado = True
        results.append((protein.name, z[0]*3, protein.sequence[0:4], sequence[z[0]*3:z[0]*3+12]))
    if not encontrado:
      results.append((protein.name, "No encontrado", "", ""))
  return results
```

[buscar palabras O(n)]

Función diseñada para obtener las palabras dentro de un árbol, en este caso se recorre de manera recursiva el árbol y se van generando las palabras a partir del valor de cada nodo recorrido.

```
def buscar_palabras(nodo, palabra):
    palabras = []
    if nodo.isFinal:
        palabras.append(palabra)
    for hijo in nodo.hijos:
        palabras += buscar_palabras(hijo, palabra + hijo.valor)
    return palabras
```

[get_codon O(1)]

Función diseñada para hacer una búsqueda directa sobre un diccionario que ha sido generado a partir del apoyo en otras funciones (Manacher, Search Proteínas). En este caso se ha reutilizado en varias secciones del código

```
def get_codon(sequence, index):
  return (diccionario_aminoacidos.buscar_palabra(sequence[index:index+3]), sequence[index:index+3])
```

[get_diff O(n)]

Función diseñada para la búsqueda lineal dentro de los archivos

```
def get_diff(sequence1, sequence2):
  results = []
  reading_frames = [0,1]

  # Por cada inicio de proteina, traducir hasta encontrar un STOP o hasta que acabe
  while len(reading_frames) > 0:
      start_index = reading_frames[0]
      end_index = reading_frames[0]

      protein = ""
      for i in range(start_index, min(len(sequence1), len(sequence2)), 3): # Itera de 3 en 3 para bu
        # codon1 = sequence1[i:i+3]
        # codon2 = sequence2[i:i+3]
        codon1, aminoacid = get_codon(sequence1, i)
        codon2, aminoacid2 = get_codon(sequence2, i)
        if codon1 != codon2:
          aparece = False
          for result in results:
            if abs(result[0] - i) <= 20:
              aparece = True
              break
          if not aparece:
            results.append((i, (aminoacid, codon1), (aminoacid2, codon2)))

      reading_frames.pop(0)

  return results
```

IV. Resultados

Usando las funciones establecidas anteriormente es que podemos obtener los siguientes resultados.

A. Índices de aparición

A cada genoma le pertenece una cadena de caracteres la cual representa sus aminoácidos, misma cadena que puede

estar presente dentro de los virus estudiados.Para conocer los índices en donde podemos encontrar cada genoma en específico se ha hecho uso de la función *StringMatchZ* la cual se apoya en la función *ZArray*..

a) Resultados para Gen M

```
print("Apariciones del gen M: ")
gen_m_indices = StringMatchZ(secuencia_wuhan, gen_m)
print(gen_m_indices)
print(secuencia_wuhan[gen_m_indices[0]:gen_m_indices[0]+12])
```

```
Apariciones del gen M:
[26522]
```

6.[Índices de aparición para el Genoma M]

b) Resultados para Gen S

```
print("Apariciones del gen S: ")
gen_s_indices = StringMatchZ(secuencia_wuhan, gen_s)
print(gen_s_indices)
print(secuencia_wuhan[gen_s_indices[0]:gen_s_indices[0]+12])
```

```
Apariciones del gen S:
[21562]
ATGTTTGTTTTT
```

7..[Índices de aparición para el Genoma S]

c) Resultados para Gen ORF 1 AB

```
print("Apariciones del gen ORF1AB: ")
gen_o_indices = StringMatchZ(secuencia_wuhan, gen_o)
print(gen_o_indices)
print(secuencia_wuhan[gen_o_indices[0]:gen_o_indices[0]+12])
```

```
Apariciones del gen ORF1AB:
[265]
ATGGAGAGCCTT
```

8.[Índices de aparición para el Genoma ORF 1 AB]

Como podemos observar solo se presenta un índice de aparición por gen, sin embargo la búsqueda continua hasta poder encontrar algún otro índice o bien , hasta terminar la secuencia.

B. Palíndromos

Otro de los análisis hechos por medio de herramientas computacionales fue el estudio de los palíndromos dentro de las cadenas de secuencia. Al presentarse un palíndromo dentro de las cadenas (en contexto del problema) nos encontramos con una repetición de aminoácidos la cual puede dar lugar para mutaciones en nuestro virus. Es por esto mismo que es necesario conocer estos palíndromos para tener presente la región en donde puede mutar un gen en específico.

Para poder encontrar estos palíndromos se ha utilizado la función de Manager para cada caso.

a) Palíndromo más largo para Gen M

```
print("Palíndromo más grande para gen M")
palindrome_m, inicio_m, fin_m = Manacher(gen_m)
print(palindrome_m, "Longitud:", len(palindrome_m))
```

```
Palíndromo más grande para gen M
CTAAAGAAATC Longitud: 11
```

9.[Resuldaos de Genoma M]

b) Palíndromo más largo para Gen S

```
print("Palíndromo más grande para gen S")
palindrome_s, inicio_s, fin_s = Manacher(gen_s)
print(palindrome_s, "Longitud:", len(palindrome_s))
```

```
Palíndromo más grande para gen S
GAATTTGTGTTTAAG Longitud: 15
```

10.[Resuldaos de Genoma S]

c) Palíndromo más largo para Gen ORF 1 AB

```
print("Palíndromo más grande para gen ORF1AB")
palindrome_o, inicio_o, fin_o = Manacher(gen_o)
print(palindrome_o, "Longitud:", len(palindrome_o))
```

```
Palíndromo más grande para gen ORF1AB
CTCAATGACTTCAGTAACTC Longitud: 20
```

11.[Resuldaos de Genoma ORF 1 AB]

C. Proteínas

Una de las necesidades de este contexto es el identificar las consecuencias de posibles mutaciones entre genomas. Más específicamente , es identificar cómo es que se presentan los cambios entre versiones de un virus durante un periodo determinado de tiempo.

Para llevar a cabo esta comprensión es necesario identificar las diferencias claves en las cadenas de arn de un virus (las 2 versiones manejadas en este trabajo), y presentarlas ante el usuario para que se tenga un reporte de tanto las mutaciones como de las diferencias significativas entre ambos.Esto es posible chacras al so de la función SearchProteionas la cual recorre el arreglo completo para identificar las posibles proteínas contenidas en el archivo

```
res = SearchProteinas(secuencia_wuhan)
for re in res:
  if re[1] != "No encontrado":
    print(re[0], ":", re[1], "- 4 amino:", re[2], "=", re[3])
  else:
    print(re[0], ":", re[1])
```

```
QHD43415_1 : 264 - 4 amino: MESL = GATGGAGAGCCT
QHD43415_2 : 804 - 4 amino: AYTR = GGCATACACTCG
QHD43415_3 : 2718 - 4 amino: APTK = TGCACCAACAAA
QHD43415_4 : 8553 - 4 amino: KIVN = TAAAATTGTTAA
QHD43415_5 : 10053 - 4 amino: SGFR = GAGTGGTTTTAG
QHD43415_6 : 10971 - 4 amino: SAVK = AAGTGCAGTGAA
QHD43415_7 : 11841 - 4 amino: SKMS = GTCTAAAATGTC
QHD43415_8 : 12090 - 4 amino: AIAS = AGCTATAGCCTC
QHD43415_9 : 12684 - 4 amino: NNEL = GAATAATGAGCT
QHD43415_10 : 13023 - 4 amino: AGNA = AGCTGGTAATGC
QHD43415_11 : No encontrado
QHD43415_12 : 16236 - 4 amino: AVGA = GCTGTTGGGGCT
QHD43415_13 : 18039 - 4 amino: AENV = GCTGAAAATGTA
QHD43415_14 : 19620 - 4 amino: SLEN = AGTTTAGAAAAT
QHD43415_15 : 20658 - 4 amino: SSQA = TCTAGTCAAGCG
QHD43416 : 21561 - 4 amino: MFVF = AATGTTTGTTTT
QHD43417 : 25392 - 4 amino: MDLF = ATGGATTTGTTT
QHD43418 : 26244 - 4 amino: MYSF = ATGTACTCATTC
QHD43419 : 26520 - 4 amino: MADS = CCATGGCAGATT
QHD43420 : 27201 - 4 amino: MFHL = ATGTTTCATCTC
QHD43421 : 27393 - 4 amino: MKII = ATGAAAATTATT
QHD43422 : 27891 - 4 amino: MKFL = ACATGAAATTTC
QHD43423 : 28272 - 4 amino: MSDN = AATGTCTGATAA
QHI42199 : 29556 - 4 amino: MGYI = GATGGGCTATAT
```

12..[Resuldaos de proteinas para secuencia wuhan]

D. Comparación

Una ve que se han listado las proteínas presentes en la versión de wuhan, se realizó una comparación con aquellas presentes en la versión Texas. Esto con la finalidad de tener una mejor comprensión sobre los puntos en lso que ha mutado el virus a lo largo de su recorrido.

Para llevar esto a cabo se hizo uso de la función get_diff.

```
[ ] res = get_diff(secuencia_wuhan, secuencia_texas)
    print("Indice\t\tWuhan\t\tTexas")
    for r in res:
      # print(r)
      # print("Diferencia en", r[0], ":", r[1][0], "("+r[1][1]+")", "vs", r[2][0], "("+r[2][1]+")")
      print(str(r[0]) + "\t\t" + str(r[1][0])+ " (" + str(r[1][1]) + ")" + "\t\t" + str(r[2][0])+ " (" + str(r[2][1])
```

```
Indice          Wuhan           Texas
8781            CCA (P)         TCA (S)
19173           GAT (D)         GCT (A)
27924           CAA (Q)         TAA (*)
28143           TAC (Y)         CAC (H)
29094           CGG (R)         TGG (W)
29880           AAA (K)         AA (~)
18601           ATG (M)         ACG (T)
18973           TTC (F)         TAC (Y)
```

13..[Resuldaos de comparación de proteinas entre las 2 versiones]

V. Contribuciones

A. Rodrigo Lopez

Desarrollo de algoritmos para alcanzar los objetivos establecidos, investigación y redacción del reporte.

B. Victor M. Escalante

Desarrollo de algoritmos para alcanzar los objetivos establecidos, investigación y redacción del reporte.

VI. Conclusiones

A. Rodrigo Lopez

Como reflexión, se me hizo interesante conocer cómo funcionan los algoritmos que vimos en clase aplicados a un campo tan importante para el desarrollo de la sociedad como es la medicina, el desarrollo de aplicaciones computacionales y algoritmos son útiles para la solución de problemas de la vida real.

B. Victor M. Escalante

En mi perspectiva , el trabajar con esta situación problema fue bastante interesante puesto que pude aprender sobre las distintas aplicaciones que tienen los conocimientos del dominio de mi carrera en otras ramas de la ingeniería.

Bibliografía

[1] J. P. Wilkinson, “Nonlinear resonant circuit devices (Patent style),” U.S. Patent 3 624 12, July 16, 1990.

[2] Enlace a colab [E1. Act Int 1]

[1] *IEEE Criteria for Class IE Electric Systems* (Standards style), IEEE Standard 308, 1969.