UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
BIOINGENIERÍA
Informática 2
Autor: Luis Miguel Gaviria C. Est.

PRÁCTICA DE LABORATORIO

Síntesis de cadenas peptídicas a partir de la transcripción del DNA.

1. OBJETIVOS

- Desarrollar un sistema de información aplicado en ingeniería Genética.
- Transcribir la información genética contenida en los cromosomas de organismos configurados a nivel celular como Eucariontes.
- Simular el proceso de anabolismo de biomoléculas proteicas a partir de precursores aminoacídicos unidos por enlaces peptídicos.
-

2. MATERIALES Y EQUIPOS

Software StarUML, Distribución Open Source Anaconda®Navigator, Python Release 3.7.5, Microsoft© Office 2019, Dispositivo Laptop integrado con procesador AMD A9 9420 RADEON R5 (3.00 GHz) – fabricante: ASUSTek Computer Inc.

3. MARCO TEÓRICO

Introducción: A mediados de la década de 1950, los investigadores habían determinado que la secuencia de bases en el ADN contiene la información que especifíca todas las proteínas que necesita la célula. Sin embargo, más de una década de intensa investigación de muchos científicos precedió a un entendimiento fundamental de cómo las células convierten la información del ADN en secuencias de aminoácidos de proteínas.

Aunque la secuencia de bases de ADN determina la secuencia de aminoácidos en polipéptidos, las células no utilizan la información en el ADN directamente. En su lugar, un ácido nucleico relacionado, el ácido ribonucleico (ARN), vincula al ADN con las proteínas. La expresión de un gen que codifica a una proteína implica primero la realización de una copia de ARN con base en la información del ADN. Esta copia de ARN es la que proporciona la información que dirige la síntesis del polipéptido (Solomon et al., 2008, p.285).

UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
BIOINGENIERÍA
Informática 2
Autor: Luis Miguel Gaviria C. Est.

Palabras clave: ADN, ARN, aminoácido, polipéptido, función recursiva, método, objeto, secuencia, estructura, base, nucleótido, proteína.

Transcripción del ADN y traducción del ARN.

El proceso mediante el cual se sintetiza la cadena de ARN se parece a la replicación del ADN en que la secuencia de bases que lo conforma está determinada por el emparejamiento de bases con una de las cadenas del ADN, la cadena codificante. Como la síntesis de ARN implica tomar la información de un tipo de ácido nucleico (ADN) y copiarlo como otro ácido nucleico (ARN), entonces este proceso se llama transcripción ("copiado").

Posterior al proceso de transcripción, la información copiada en el ARNm se utiliza para especificar la secuencia de aminoácidos de un polipéptido. Este proceso se llama traducción porque implica la conversión del "lenguaje de ácido nucleico" en la molécula de ARNm al "lenguaje de aminoácido" de la proteína.

En la traducción de las instrucciones genéticas para formar un polipéptido, cada secuencia de tres bases nucleótidas consecutivas en el ARNm, llamada codón, especifíca un aminoácido. Por ejemplo, un codón que corresponde al aminoácido fenilalanina es ¬UUC¬.

Simulación: Basando el desarrollo de un banco de datos proteicos a partir de la teoría expuesta, se busca a través de un algoritmo ensamblado por secciones funcionales independientes. Crear, validar, secuenciar y enlazar como resultado final, los diferentes elementos aminoacídicos derivados de la simulación de los procesos de transcripción y traducción de la información genética representada en una estructura de datos de tipo *String* (str).

Para tal propósito, la implementación de las diferentes estructuras de datos, así como las diferentes estructuras de control de flujo de ejecución (instrucciones *switch*, condicionales, bucles *while* y *for*), estarán implicadas en la codificación del algoritmo. Las interacciones entre las diferentes entidades del proceso estarán regidas por el paradigma de *programación orientado a objetos* (*POO*) el cual facilita la simulación y el entendimiento de las diferentes relaciones que existen entre las entidades que emulan el contexto real en el que ocurre la síntesis de cadenas polipeptídicas a nivel celular.

Al igual que las biomoléculas que intervienen en el proceso de expresión génica, las abstracciones definidas para representarlas estarán dotadas de las funcionalidades básicas propias de estas biomoléculas considerando el alcance y la delimitación que tendrá el programa; como uno de los objetivos propuestos es simular este proceso de transcripción y traducción génica en particular, serán excluidas muchas funcionalidades que los elementos reales implicados en este subproceso pueden aportar para la realización de otras funciones precursoras o alternas del ciclo celular.

Alcance del programa: El algoritmo permitirá al investigador o administrador del banco de proteínas, visualizar toda la información relacionada a una síntesis en particular, el ADN codificante, la cadena de ARN transcrita, la secuencia de codones y la cadena peptídica resultante al igual que el código identificador de la proteína serán extraídos de una estructura de datos denominada matriz que en términos del lenguaje de codificación empleado es una lista que contiene listas.

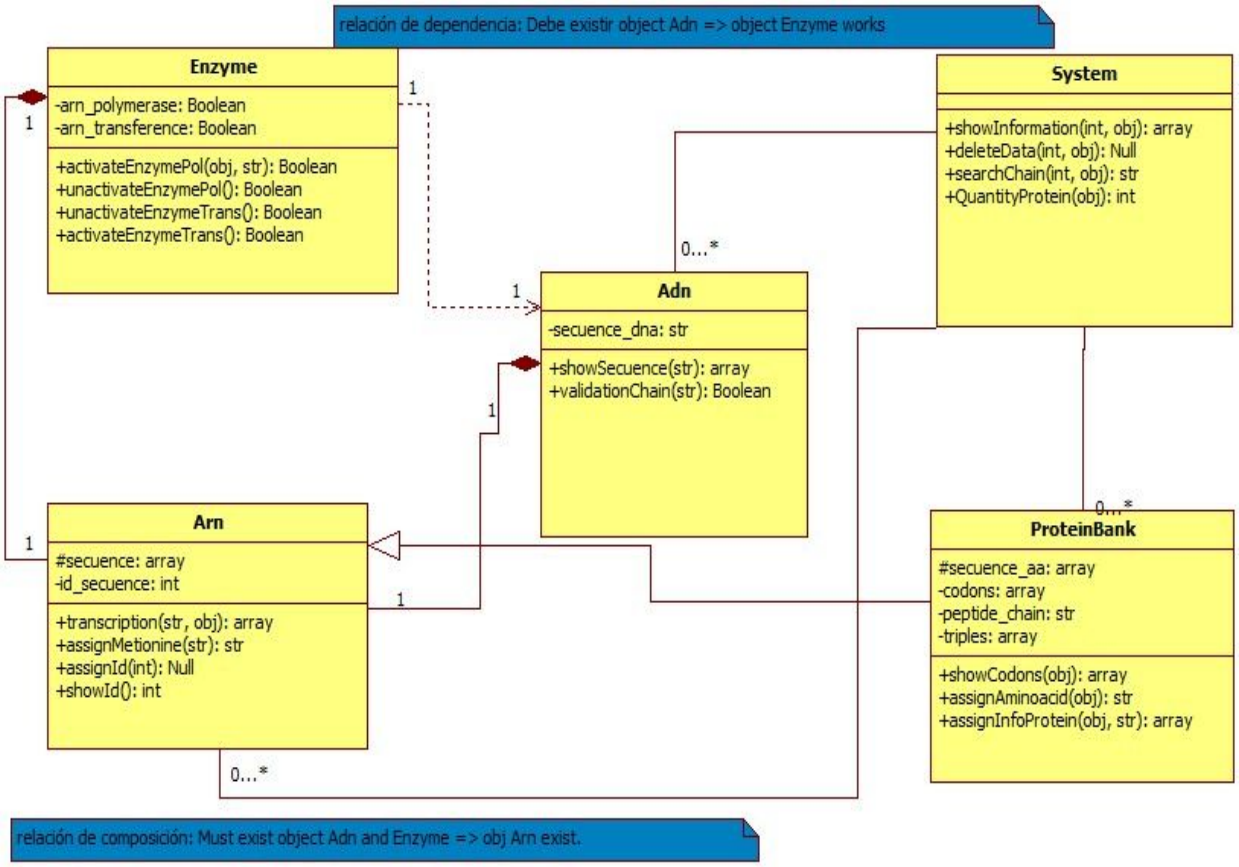
Adicional a la función principal de visualización de datos, el programa estará instruido de forma opcional para modificar la estructura primaria de una proteína. Esto es, la modificación estructural de la cadena de aminoácidos lo cual es de utilidad si se pretende alterar la función biológica de la biomolécula; esta funcionalidad será ejecutada según el criterio del investigador, el cual deberá ingresar el código de la proteína y la posición del aminoácido en la secuencia a partir del cual efectuará el corte de la cadena peptídica.

Finalmente, se incorporan métodos para eliminar completamente una estructura de datos relacionada a una proteína, consultar a través del código una cadena transcripta de ARN y obtener un registro de la cantidad de macromoléculas almacenadas en el banco de datos.

Implementación y diseño: Las abstracciones y relaciones diseñadas para simular el proceso de expresión génica, son presentadas a través de un Diagrama de clases con las anotaciones necesarias y tipos de relaciones debidamente diferenciadas con el fin de proveer la lógica de las interacciones que existen entre los diferentes objetos del diagrama. Por ejemplo, la dependencia que existe de la clase *Enzyme* sobre la clase *Adn,* la cual necesita de una instancia de ella para que pueda ejecutar sus métodos dirigidos a permitir el proceso de transcripción; sin implicar esto, que la clase *Enzyme* no pueda existir sin la clase Adn. Por su parte, la clase Arn, sí tiene una relación de composición absoluta 1…1 con las clases de Enzyme y Adn, esto

significa que el ARN no podrá existir sin que exista el ADN y tampoco sin que exista la clase Enzyme.

Toda la sustentación y explicación detallada de las diferentes funcionalidades, será expuesta con el código del algoritmo el cual estará comentado e indentado en el paso a paso.



UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
BIOINGENIERÍA
Informática 2
Autor: Luis Miguel Gaviria C. Est.

4. PROCEDIMIENTO

```
# -*- coding: utf-8 -*-
"""
Created on Sun Jul 25 16:27:02 2019

@author: Luis Miguel Gaviria
"""
class Adn(object):

    def __init__(self):  #constructor con un unico atributo para la clase, el cual
                         #contendrá la secuencia inicial de nucleotidos.
        self.__secuence = ''

    def showSecuence(self, chain):
        self.__secuence = list(chain.upper())

        return self.__secuence  # retorna como dato, la lista de nucleotidos repre-
                                #sentados en Letras mayusculas por convención.

    def validationChain(self, chain):  #método fundamental para validar que la secuencia de
                                       #bases sea correcta respecto a la convención usada para bases.
        for k in chain:
            if k != 'a' and k != 't' and k != 'c' and k != 'g':

                print('La secuencia de datos no corresponde es su totalidad con los nucleótidos correctos' + '\n')
                print('--------------------------------------------------------------------------------')
                data = input("Ingresa una secuencia aleatoria de bases nitrogenadas " + "|A - T - C - G|: ")
                self.validationChain(data)  #función recursiva que se invoca así misma para ejecutarse nueva/
            else:
                pass
        print('-----------------------------------', '\n', 'La información genética ha sido validada', '\n-----------------------------------')

        return True
    #-------------------------------------------------------------------
class Enzyme(object):

    def __init__(self):  #constructor con dos unicos atributos para la clase, los cuales
                         #definirán Los estados de las enzimas on/off, inicial/ estarán inactivas .
        self.arn_polymerase = False
        self.arn_transference = False

    def activateEnzymePol(self, obj, chain):  #Se activa la enzima ARN Polimerasa si existe una secuencia de adn(obj) y es valida.
                                              #También deberá estar activa para permitir la transcripción a ARNm
        if len(obj.showSecuence(chain)) != 0:
            self.arn_polymerase = True

            return self.arn_polymerase  #La enzima cambia de estado y con ello es posible iniciar la transcripción a ARNm.

        else:
            return self.arn_polymerase  #se mantendrá el estado en caso de que la secuencia adn no sea valida.
    #-----------------------------------
    def unactivateEnzymePol(self):  #fundamentAL para activar la enzima de ARNt e iniciar la traducción.

        if self.arn_polymerase == True:

            self.arn_polymerase = False

        return True  #Fundamental no tener return booleano dentro de bucles o condicionales.
#         else:
#             return None
    #-----------------------------------
    def activateEnzymeTrans(self):  #La asociación de aminoácidos con los respectivos codones tiene lugar solo si la enzima trans
                                    #se activa

        if self.unactivateEnzymePol() == True:
            self.arn_transference = True

        return self.arn_transference
#         else:
#             return None
    #-----------------------------------
    def unactivateEnzymeTrans(self):  #Desactiva enzyma cuando todo proceso halla concluido,

        self.arn_transference = False
        print('.....................................................................\n',
              'El proceso de traducción génica ha sido completado exitosamente.')

        #return self.activateEnzymeTrans()
#-------------------------------------
class Arn(object):

    def __init__(self):

        self.secuence_arn = []
        self.id_secuence = 0
    #--------------------------
    def transcription(self, chain, obj):  #método ejecuta proceso de transcripción adn -> arn
        #retorna el método publico de la clase Adn al atributo de la clase Arn para iniciar proceso.
        self.secuence_arn = obj.showSecuence(chain)
        for k in self.secuence_arn:

            if k == 'T':
                self.secuence_arn[self.secuence_arn.index(k)] = 'U'

            else:
                pass

        return self.secuence_arn
```

UNIVERSIDAD DE ANTIOQUIA

UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
BIOINGENIERÍA
Informática 2
Autor: Luis Miguel Gaviria C. Est.

```
    #-------------------------------------------------------------------
    def assignMetionine(self, met = 'AUG'):  #Codon de inicio del proceso de traducción a aminoácidos, debe agregarse al Arn
        i = 0
        for k in met:
            self.secuence_arn.insert(i, k)
            i+= 1

        return self.secuence_arn
    #-------------------------------------------------------------------
    def assignId(self, num):  #asigna un número de id a la secuencia de ARN

        self.id_secuence = num

    def showId(self):   #método que será invocado para retornar id de la cadena arn

        return self.id_secuence

    def validationType(self, num, obj):  #método para validar tipo número del id de arn

        try: #control de excepciones o errores de ejecución para tipo de dato, validación

            if type(num) == str:
                num = int(num)

        except ValueError:

            print('Debes identificar la secuencia de Arn con una secuencia numérica' + '\n')
            print('-------------------------------------------------------------------------')
            n = int(input("Ingresa nuevamente un número de identificación para la secuencia de ARN: "))
            self.validationType(n, obj)  #función recursiva que se invoca así misma para ejecutarse nueva/

        if str(num) in obj.secuence_aa.keys():

            print('El número de identificación ingresado ya existe en nuestra base de datos' + '\n')
            print('-------------------------------------------------------------------------')
            n = int(input("Ingresa nuevamente un número de identificación diferente al anterior para continuar: "))
            self.validationType(n, obj)  #función recursiva que se invoca así misma para ejecutarse nueva/


        return True
#-----------------------------------------------------------------------------------
class ProteinBank(Arn): # Clase que contine diccionario de codones asociados con aminoacidos
                        #Clase con métodos para secuenciar la cadena arn en tripletas y ligar con aa.
    def __init__(self):

        Arn.__init__(self)  ##Herencia de atributos de la clase Arn - Concepto de Herencia
        self.peptide = ''
        self.secuence_aa = {}
        self.triples = []
        self.codons = ({"GCU":"Ala","GCC":"Ala","GCA":"Ala","GCG":"Ala","GAU":"Asp","GAC":"Asp","UGU":"Cys","UGC":"Cys",
                        "CGU":"Arg","CGC":"Arg","CGA":"Arg","GGU":"Gly","GGC":"Gly","GGA":"Gly","GGG":"Gly","CAU":"His",
                        "CAC":"His","AAA":"Lys","AAG":"Lys","UUU":"Phe","UUC":"Phe","CCU":"Pro","CCA":"Pro","CCC":"Pro",
                        "CCG":"Pro","UCU":"Ser","UCC":"Ser","UCA":"Ser","GUU":"Val","GUC":"Val","GUA":"Val","GUG":"Val",
                        "ACU":"Thy","ACC":"Thy","ACA":"Thy","ACG":"Thy","UAU":"Tyr","UAC":"Tyr","AUG":"Met"})

    def showCodons(self, obj): #método para depurar la cadena de ARN y agrupar las tripletas de bases, obj de tipo Arn

        self.id_secuence = obj.id_secuence  #esta función solo se ejecuta si la enzima self.__arn_transference se encuentra activa.
        self.triples = obj.secuence_arn[:] #copia por slicing de la cadena de transcripción Luego de ser agregado el codon Metionina
        if len(self.triples) % 3 == 2:  #instrucciones de decisión para verificar Longitud con multiplicidad
            del self.triples[-2:]       # exacta de 3 y eliminar nucleótidos finales.

        elif len(self.triples) % 3 == 1:
            del self.triples[-1]

        string = ''
        while len(self.triples[0]) == 1:  #Bucle para cortar el arn y transformarlo en codones o tripletas
                                          #se redefine la lista contenedora de las bases por los codones derivados
            for k in range(3):            #de la secuencia de arn. Una vez la longitud del primer elemento es != 1
                string = string + self.triples[k] #se termina el ciclo de cortes y la lista es redefinida a vector de codones

            del self.triples[:3]  #
            self.triples.append(string)
            string = ''

        return self.triples
    #-------------------------------------------------------------------------------
    def assignAminoacid(self, obj): #parametro de tipo Enzyme para asignar aa si enzima transferasa se encuentra activa

        array = self.triples[:]  #variable local que hace un slicing del retorno de la función anterior

        if obj.activateEnzymeTrans() == True:

            for w in array:
                if w in self.codons.keys(): #recorre el vector de claves del diccionario para verificar existencia

                    self.peptide = self.peptide + self.codons[w] + '--'

                else:
                    continue

            return self.peptide
        #-------------------------------------------------------------------
    def assignInfoProtein(self, arn, adn, data):  #-objetos por parametros.

        self.secuence_aa[arn.showId()] = [adn.showSecuence(data), arn.secuence_arn, self.triples, self.peptide]
```

Autor: Luis Miguel Gaviria C. Est.

```
        #diccionario que almacenará los pares id_proteína : datosobj.secuence_aa
#-------------------------------------------------------------------------------------------------------------------------------------
class System(object):

    def showInformation(self, num, obj):

        num = str(num)
        if num in obj.secuence_aa.keys():

            print ('Código de secuencia genética -', str(obj.showId())) #muestra en pantalla el código luego la información completa.
            for k in obj.secuence_aa[num]:
                print('----------------------------------------------------------' + '\n' + str(k))

        elif type(num) != int:

            print('La identificación debe corresponder con un dato unicamente numerico' + '\n')
            print('-------------------------------------------------------------------------------')
            n = input("Ingresa el código de identificación para la proteína : ")
            self.showInformation(n, obj)

        else:
            print('-------------------------------------------------------------------------------')
            print('El número que ingresaste para identificar la proteína no está asociado en nuestro banco de datos')
            print('-------------------------------------------------------------------------------' + '\n')
            m = int(input("Ingresa de nuevo el número identificador de la proteína para consultar los datos: "))
            self.showInformation(m, obj)  #función recursiva que se invoca así misma para ejecutarse nuevamente
            #-----------------------------------------

    def searchChain(self, num, obj):

        num = str(num)
        if num in obj.secuence_aa.keys():

            k = obj.secuence_aa[num]  #accede a los datos asociados con el código
            return k[3]  #retorna el elemento 4 de la lista de datos obtenidos al buscar por la clave de proteína

        else:
            print('-------------------------------------------------------------------------------')
            print('El número que ingresaste para identificar la proteína no está asociado en nuestro banco de datos')
            print('-------------------------------------------------------------------------------' + '\n')
            z = int(input("Ingresa de nuevo el identificador de la proteína para consultar la secuencia peptidica: "))
            self.searchChain(z, obj)  #función recursiva que se invoca así misma para ejecutarse nuevamente
            #-----------------------------------------

    def quantityProtein(self, obj):

        print ('----------------------------------------------------' + '\n',
               'El banco de datos actualmente contiene ' + str(len(obj.secuence_aa.keys())) + ' proteínas codificadas.')

    def deleteData(self, num, obj):

        num = str(num)
        if num in obj.secuence_aa.keys():
            del obj.secuence_aa[num]

            return ('.....................................................................\n',
                    'Los datos de la proteína han sido eliminados del sistema.')
        else:
            return 'El código de identificación no existe en la base de datos'


#-------------------------------------------------------------------------------------

def main():

    var, adn, enzyme, arn, protein, sys = True, Adn(), Enzyme(), Arn(), ProteinBank(), System() # Se instancian todos los objetos.
    print('-----------------------------------', '\n', 'Simulador de síntesis de aminoácidos', '\n-----------------------------------')
    while var == True:

        data = input('Ingresa una secuencia de nucleotidos de Adn para transcribir y traducir la información genética: ')#
        if adn.validationChain(data) == True:

            if enzyme.activateEnzymePol(adn, data) == True:

                arn.transcription(data, adn)  #adn.showSecuence(data)
                arn.assignMetionine()
                num = input('-----------------------------------------\nAsigna a la secuencia de Arn un código numérico: ')
                arn.validationType(num, protein)
                enzyme.unactivateEnzymePol()  #Inactiva la enzima de arn polimerasa para activar transferasa

                if arn.validationType(num, protein) == True:

                    arn.assignId(num)
                    enzyme.activateEnzymeTrans()  #Enzima que permite asociar los aa con los codones.
                    protein.showCodons(arn)       #se genera el vector de codones.
                    protein.assignAminoacid(enzyme)     #se asocian los codones con aminoácidos
                    protein.assignInfoProtein(arn, adn, data)  #Se almacenan todos los datos en diccionario.
                    enzyme.unactivateEnzymeTrans()  #Inactiva enzima transferasa.
                else:

                    pass

            else:
                continue

        print ('----------------------------\n' + 'Opciones de Consulta de datos' +'\n----------------------------------\n',
               '0 - Iniciar una nueva síntesis peptídica \n 1 - Consultar secuencias Adn, Arn y péptidos \n',
               '2 - Consultar secuencia de aminoácidos sintetizados\n',
               '3 - Verificar la cantidad de registros en el banco de datos\n 4 - Eliminar bloque de datos de una proteína\n',
               '5 - Finalizar consulta de datos\n-------------------------------------------------')
```

UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
BIOINGENIERÍA
Informática 2
Autor: Luis Miguel Gaviria C. Est.

```
        main = int(input('Ingresa una opción numerica de acuerdo al menú presentado anteriormente: '))
        if main == 0:
            continue
        while True:
            if main == 1:
                num = int(input("Ingresa el número identificador de la proteína para consultar los datos: "))
                sys.showInformation(num, protein)
            elif main == 2:
                num = int(input("Ingresa el número identificador de la proteína para consultar la síntesis de péptidos: "))
                print ('-----------------------------------------------------------------')
                print (sys.searchChain(num, protein))
            elif main == 3:
                print(sys.quantityProtein(protein))
            elif main == 4:
                num = int(input("Ingresa el número identificador de la proteína para descartar los datos del sistema :"))
                print (sys.deleteData(num, protein))
            elif main == 5:
                var = False
                break
            main = int(input('Si deseas ejecutar una nueva consulta escoge una opción de nuevo: '))
        var = False
# print('-----------------------------', '\n', ,  '\n-----------------------------')
if __name__ == '__main__':
    main()
```

5. Referencias Bibliográficas.

- Python Software Foundation. (2017). The Python Standard Library. 25 Julio, 2019, de Python Software Foundation Sitio web: https://docs.python.org/3/library/index.html

- Mark Pilgrim. (2009). Dive into Python. Edición 2nd: Apress.