

Alineamiento de secuencias

Implementación de código de python en R

 Ronald Rivas¹, Fabián Olivella¹, and Luis Carmona¹

¹ Universidad Nacional de Colombia; Escuela de pregrados. Estudiantes de ingeniería biológica.

Resumen

La secuenciación y alineamiento del ADN toman importancia en la mayoría de procesos biológicos que son llevados a cabo en un laboratorio, es por esto que el tener un programa que permite tener el mejor alineamiento posible de las secuencias ayudaría a mejorar no solo en cuanto eficiencia, sino que también en versatilidad el desarrollo de dichos procesos. Por otra parte, se sabe que el lenguaje R es uno de los más usados en la academia para el manejo de datos además de Python. En este proyecto se tiene como objetivo hacer una comparación entre los dos lenguajes al intentar implementar un algoritmo que ayuda a dar solución a un problema de secuenciación, desde Python a R, para así observar las dificultades que se puedan presentar durante la realización de dicha implementación.

Palabras clave: Implementación, Alineamiento, Secuencias, Programación; Lenguaje R; Lenguaje Python, análisis.

Introducción

El ácido desoxirribonucleico (ADN) es la molécula que posee el material genético y la información de todos los seres vivos, conteniendo las instrucciones para que los seres vivos puedan reproducirse y mantener su vida. El ADN se organiza en forma de código el cual está compuesto por 4 moléculas conocidas como bases, citosina (C), timina (T), guanina (G) y adenina (A) ([MedlinePlus](#), 2021).

La secuenciación del ADN consiste entonces en determinar el orden de las bases que componen al ADN (timina, guanina, citosina, adenina), esta información es utilizada para determinar qué fragmentos de ADN corresponden a genes y cuáles desactivan o activan los mismos, pudiendo incluso detectar enfermedades ([NIH](#), 2019).

Como se ha dicho anteriormente estas secuencias de ADN pueden indicar las funciones de las proteínas dentro de los seres vivos. Por un lado, cuando más parecidas sean estas secuencias más parecidas serán sus funciones, mientras que por otro lado, las secuencias que correspondan a un gen específico serán más diferentes en función de qué tan alejadas filogenéticamente están las especies. Pero para poder identificar estas similitudes o parentescos primeramente las secuencias deben ser alineadas ([Genomics](#), n.d.).

Debido a esto surge entonces “El problema de alineación de secuencias” el cual es uno de los problemas biológicos fundamentales que consiste en encontrar similitudes y diferencias entre 2 secuencias de aminoácidos (ADN en este caso) con el propósito de comparar un gen, encontrar dominios funcionales y encontrar posiciones homólogas, permitiendo a su vez resolver dudas con respecto a la evolución y el desarrollo ([GeeksforGeeks](#), 2022).

Este problema ya ha sido abordado mediante la programación utilizando diversos lenguajes de programación y el algoritmo de programación dinámica presentado por Saul B. Needleman y Christian D. Wunsch en el año 1970, en donde se introducen huecos en las cadenas para de esta forma igualar las longitudes ([GeeksforGeeks](#), 2022) ([Lekberg](#), 2020).

El objetivo de este informe es entonces realizar una comparación entre los lenguajes de programación Python y R, abordando problemáticas de índole biológica como “El problema

de alineación de secuencias” de tal forma que se puedan observar las diferencias entre la realización, implementación y solución de problemas en cada lenguaje.

Planteamiento del Problema

Se busca implementar el código que presenta una solución al problema de alineamiento de secuencias propuesto por John Lekberg realizado en lenguaje python al lenguaje de programación R. Primero, el autor John Lekberg, realizó una solución con la ayuda de una técnica conocida como “búsqueda fuerza bruta” y a partir de esta, depurando el código generado por la misma, implementó una solución en Python que resuelve el problema mencionado anteriormente.

Debido a la extensión del script de John Lekberg, en este proyecto solo se busca implementar la primera función presente en el cuaderno de colab, dicha toma dos rangos de índices e itera sobre todas las alineaciones posibles.

```
1 from collections import deque
2
3
4 def all_alignments(x, y):
5     """Devuelve un iterable de todos los alineamientos de dos
6     secuencias.
7
8     x, y -- Secuencias.
9     """
10
11     def F(x, y):
12         """Una función de ayuda que construye recursivamente las
13         alineaciones.
14
15         x, y -- Índices de secuencias para los originales x e y.
16         """
17
18         if len(x) == 0 and len(y) == 0:
19             yield deque()
20
21         escenarios = []
22         if len(x) > 0 and len(y) > 0:
23             escenarios.append((x[0], x[1:], y[0], y[1:]))
24         if len(x) > 0:
25             escenarios.append((x[0], x[1:], None, y))
26         if len(y) > 0:
27             escenarios.append((None, x, y[0], y[1:]))
28
29         # NOTA: "xh" y "xt" significan "cabeza x" y "cola x",
30         # siendo "cabeza" la parte delantera de la secuencia, y
31         # "cola" es el resto de la secuencia. Del mismo modo, para
32         # "yh" e "yt".
33         for xh, xt, yh, yt in escenarios:
34             for alignment in F(xt, yt):
35                 alignment.appendleft((xh, yh))
36             yield alignment
37
38     alignments = F(range(len(x)), range(len(y)))
39     return map(list, alignments)
```

Figura 1. Función en Python

Proceso de Implementación

Para poder implementar el código de John Lekberg en R de una manera amena, se empleó una técnica recursiva conocida como DAC por sus siglas en inglés (Divide And Conquer) para dividir el problema a resolver en varios problemas pequeños.

Lo primero a intentar resolver fue la función anidada $F(x,y)$. Dentro de esta función fue dividida en dos partes, la primera que consiste en, a partir de los datos introducidos como “x” y “y”, con la ayuda de los condicionales, generar listas de caracteres y guardarlas en otra lista llamada “escenarios” (se comprobó aislando las líneas de código e imprimiendo “escenarios” con el locus de la liste creada tras cada condicional y al final del todo).

```

1 x = "CAT"
2 y = "CT"
3
4 escenarios = []
5
6 if len(x) > 0 and len(y) > 0:
7     escenarios.append((x[0], x[1:], y[0], y[1:]))
8     print(scenarios[0])
9 if len(x) > 0:
10    escenarios.append((x[0], x[1:], None, y))
11    print(scenarios[1])
12 if len(y) > 0:
13    escenarios.append((None, x, y[0], y[1:]))
14    print(scenarios[2])
15
16 print()
17 print(scenarios)

```

```

('C', 'AT', 'C', 'T')
('C', 'AT', None, 'CT')
(None, 'CAT', 'C', 'T')

[('C', 'AT', 'C', 'T'), ('C', 'AT', None, 'CT'), (None, 'CAT', 'C', 'T')]

```

Ahora bien, la implementación de este fragmento en R es, a priori, más extensa en cuanto a líneas de código.

Para el primer intento se hizo uso de la función “nchar”, que cuenta el número de caracteres dentro de una variable o que le sean directamente ingresadas, también se hizo uso de la función “paste0” con el fin de juntar caracteres y por último, se hizo uso de la función “append” que vendría siendo el homólogo a la misma función presente en python.

El primer intento fue el siguiente:

```

1 x = "CAT"
2 y = "CT"
3
4 escenarios = data.frame()
5
6 if (nchar(x) > 0 && nchar(y) > 0){
7     p1 = c(x[1],paste0(x[2],x[3]),y[1],y[2])
8     escenarios = append(scenarios, list(p1))
9     print(scenarios[[1]])

```

Sin embargo, al momento de imprimir “scenarios”, no se obtuvo un resultado, esto puede ser debido a que R no toma cada carácter como un elemento exactamente como Python.

Para el segundo intento, en lugar de asignar caracteres en las variables, se optó por guardar en las mismas vectores con cada letra separada. También se realizó el cambio de la función “nchar” por la función “length” para así poder tener el largo del vector.

```

1 x = c("C", "A", "T")
2 y = c("C", "T")
3
4 scenarios = data.frame()
5
6 if (length(x) > 0 && length(y) > 0){
7     p1 = c(x[1], paste0(x[2], x[3]), y[1], y[2])
8     scenarios = append(scenarios, list(p1))
9     print(scenarios[[1]])
10 }

```

Al imprimir “scenarios” el resultado fue el esperado, se obtuvo la siguiente lista:

[1] “C”, “AT”, “C”, “T”

Por lo que se termina el código y se ejecuta para comprobar los resultados:

```

1 x = c("C", "A", "T")
2 y = c("C", "T")
3
4 scenarios = data.frame()
5
6 if (length(x) > 0 && length(y) > 0){
7     p1 = c(x[1], paste0(x[2], x[3]), y[1], y[2])
8     scenarios = append(scenarios, list(p1))
9     print(scenarios[[1]])
10 }
11 if (length(x) > 0){
12     p2 = c(x[1], paste0(x[2], x[3]), "None", paste0(y[1], y[2]))
13     scenarios = append(scenarios, list(p2))
14     print(scenarios[[2]])
15 }
16 if (length(y) > 0){
17     p3 = c("None", paste0(x[1], x[2], x[3]), y[1], y[2])
18     scenarios = append(scenarios, list(p3))
19     print(scenarios[[3]])
20 }
21
22 print(scenarios)

```

De la que se obtuvo la siguiente salida:

```

[1] "C", "AT", "C", "T"
[1] "C", "AT", "None", "T"
[1] "None", "AT", "C", "T"

[[1]] "C", "AT", "C", "T"
[[2]] "C", "AT", "None", "T"
[[3]] "None", "AT", "C", "T"

```

En python se pueden crear bucles for con varias variables temporales que permiten acceder a los elementos de tuplas que se encuentran anidadas en una lista. Para la segunda parte de la implementación al lenguaje R debíamos hallar una forma para hacer lo mismo. Debido a que en el lenguaje R solo se puede poner una variable temporal, la forma más cercana a lo que se hizo en python que se nos ocurrió para dicho proceso fue utilizando bucles for anidados que permitan acceder a las listas o vectores de una forma parecida a lo realizado por Lekberg en python. Esto es de la forma:

Bucle for con varias variables aislado (Python):



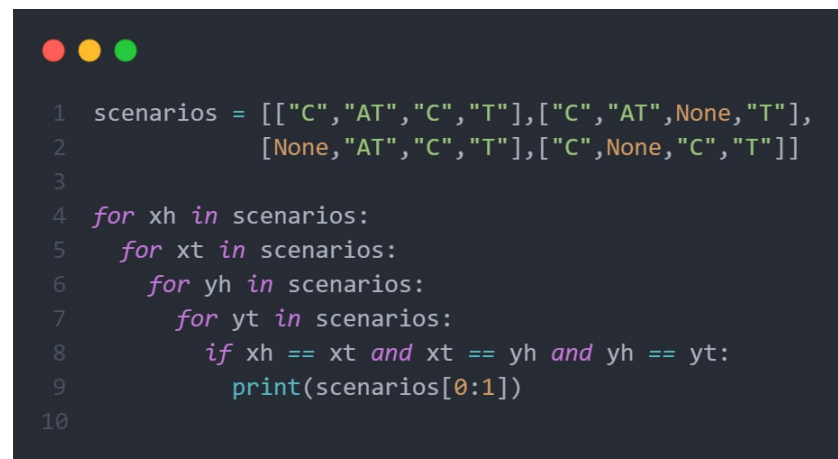
```

1  escenarios = [{"C","AT","C","T"},{"C","AT",None,"T"},
2              [None,"AT","C","T"],{"C",None,"C","T"}]
3
4  def f():
5      for xh,xt,yh,yt in escenarios:
6          print(scenarios[0:1])
7      return
8
9  f()

```

Figura 2. Bucle "For" con multiples variables.

Bucle for con varios for anidados para cada variable iterativa + condición (Python):



```

1  escenarios = [{"C","AT","C","T"},{"C","AT",None,"T"},
2              [None,"AT","C","T"],{"C",None,"C","T"}]
3
4  for xh in escenarios:
5      for xt in escenarios:
6          for yh in escenarios:
7              for yt in escenarios:
8                  if xh == xt and xt == yh and yh == yt:
9                      print(scenarios[0:1])
10

```

Figura 3. Bucle "For" con "For" anidados en Python.

Bucle for con varios for anidados para cada variable iterativa + condición (R):

```
1 for (xh in escenarios[[2]]){
2   for (xt in xh){
3     for (yh in xt){
4       for (yt in yh){
5         if (xh == xt && xt == yh && yh == yt){
6           for (alignment in ff(xt, yt)){
7             append(alignment, c(xh, yh), after = 0)
8             #print(scenarios[[1]]) # Aquí va el alignment
9           }
10        }
11      }
12    }
13  }
14 }
```

Figura 4. Bucle "For" con "For" anidados en R.

Código completo implementado en R, aún disfuncional:

```
1 x = c("C", "A", "T")
2 y = c("C", "T")
3
4 allAlignment = function(x,y){
5   "
6   Funcion principal
7   "
8   ff = function(x,y){
9     "
10    Funcion anidada
11    "
12    #if (length(x) == 0 && length(y) == 0){
13    #Aquí va el yield deque()
14    }
15    escenarios = data.frame()
16    if (length(x) > 0 && length(y) > 0){
17      p1 = c(x[1], paste0(x[2], x[3]), y[1], y[2])
18      escenarios = append(scenarios, list(p1))
19    }
20    if (length(x) > 0){
21      p2 = c(x[1], paste0(x[2],x[3]), "None", paste0(y[1],y[2]))
22      escenarios = append(scenarios, list(p2))
23    }
24    if (length(y) > 0){
25      p3 = c("None",paste0(x[1], x[2], x[3]), y[1], y[2])
26      escenarios = append(scenarios, list(p3))
27    }
28
29    for (xh in escenarios[[2]]){
30      for (xt in xh){
31        for (yh in xt){
32          for (yt in yh){
33            if (xh == xt && xt == yh && yh == yt){
34              for (alignment in ff(xt, yt)){
35                append(alignment, c(xh, yh), after = 0)
36                #print(scenarios[[1]]) # Aquí va el alignment
37              }
38            }
39          }
40        }
41      }
42    }
43    alignments
44    return()
45  }
46  alignments = ff(seq(length(x)), seq(length(y)))
47  return(ff(alignments))
48 }
```

Figura 5. Código implementado en R (Disfuncional).

Conclusión

El “problema de alineación de secuencias” ha demostrado tener una solución de alta complejidad debido a que se hacen uso de estructuras como matrices, varios for anidados y funciones dentro de funciones, que pueden generar confusiones para aquellos que empiezan en la programación.

Ambos lenguajes de programación poseen sus pros y sus contras, python es mucho más fácil de aprender a manejar que R e inclusive este tiene mucho mas librerias de donde elegir a diferencia de R, aunque por otro lado R es mucho más utilizado en “Data-science”, analisis de datos y visualización de los mismos.

Es complicado implementar el código de alineación de secuencias en R por las siguientes razones: Algunos vectores tienden a comportarse diferente en R pues todos son solo eso, vectores. A diferencia de Python que tiene varias formas de almacenar variables, así como también, algunas funciones como el Yield() y el Deque(), funcionan de una manera diferente en ambos lenguajes, y aunque comparten similitudes (en el caso del Deque) esta necesita de otros argumentos para funcionar pero esto no quiere decir que se obtenga el resultado esperado. Por último, Python es más flexible que R y tiende menos a colgarse al momento de ejecutar bucles, o variables iterativas sin guardarlas anteriormente (como al usar el Yield).

References

- GeeksforGeeks.** (2022). *Sequence alignment problem*. Retrieved from <https://www.geeksforgeeks.org/sequence-alignment-problem/>
- Genomics, B. .** (n.d.). *Alineamiento de secuencias*. Retrieved from https://bioinf.comav.upv.es/courses/intro_bioinf/alineamientos.html
- Lekberg, J.** (2020, 10). *Solving the sequence alignment problem in python*. Retrieved from <https://johnlekberg.com/blog/2020-10-25-seq-align.html>
- MedlinePlus.** (2021). *¿qué es el adn?* Retrieved from <https://medlineplus.gov/spanish/genetica/entender/basica/adn/>
- NIH.** (2019). *¿qué es la secuenciación del adn?* Retrieved from <https://www.genome.gov/es/about-genomics/fact-sheets/Secuenciacion-del-ADN>