Implementación del algoritmo InexRecur en Python y comparación con implementación en R.*

Ignacio Amat Hernández Ángela LLopis Hernández Estrella Ballester Hoyo

4 de mayo de 2020

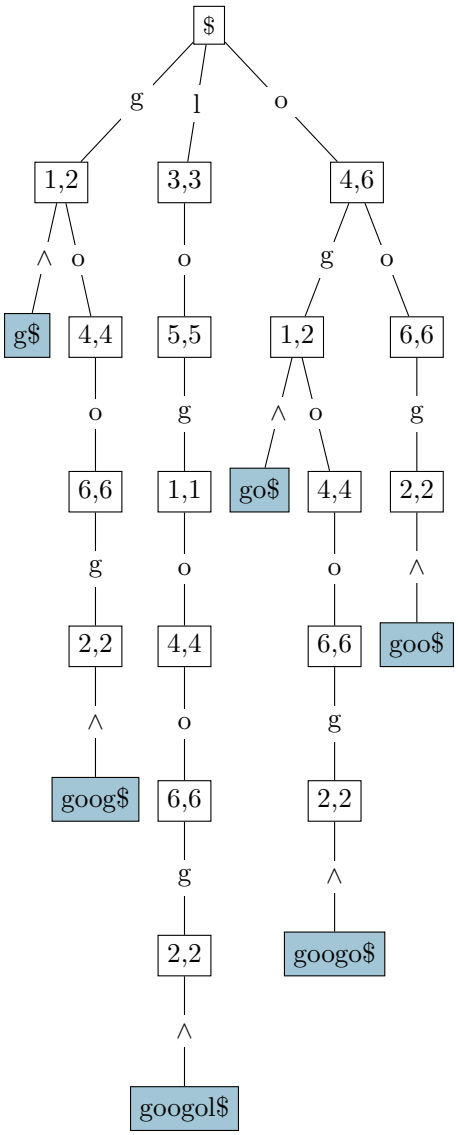

Figura 1: Árbol de prefijos con intervalos SA de la palabra "googol$".

*Grado en Ingeniería Biomédica, Escuela Técnica Superior de Ingenieros Industriales, Valencia, España.

Índice

Índice de figuras

Índice de cuadros

Listings

Resumen

Debido a la aparición de nuevas técnicas de secuenciación masiva (NGS) en la bioinformática, existen actualmente numerosas herramientas para la secuenciación. Algunas de ellas han sido diseñadas para resolver el problema de alineamiento, entre ellas el Burrows-Wheeler Aligner (BWA), un alineador diseñado para alinear de secuencias a partir de un genoma de referencia [1]. BWA se basa en la transformada de Burrows-Wheeler (BWT) y tiene la capacidad de realizar búsquedas inexactas (permitiendo huecos y errores) de forma recursiva mediante el método InexRecur [2].

Hemos programado el algoritmo de recursión InexRecur en Python para el alineamiento de secuencias cortas con secuencias de referencia de gran tamaño y hemos realizado diversas pruebas comparando con la implementación del mismo algoritmo en R. Hemos obtenido las trazas en Python, R y C, los tiempos de CPU, la memoria requerida y los árboles de prefijos.

Los resultados obtenidos muestran que la implementación de la función de búsqueda recursiva inexacta en Python proporciona un menor tiempo de procesamiento y menor uso de memoria, además de poseer una sintaxis más intuitiva para el usuario en comparación con R.

1. Introducción

El campo de la bioinformática es una rama de la informática combinada con la biología, la química, la estadística y muchos otros ámbitos que trata de recopilar, almacenar, analizar, manipular, etc. datos de carácter biológico con el fin de comprender el funcionamiento de los organismos vivos, su origen y poder hallar así principios universales relativos a las ciencias de la vida [1]. Es un ámbito de gran interés para la industria científica pero que ha de manejar un gran número de datos (normalmente una máquina de secuenciación Illumina/Solexa puede producir de 50 a 200 millones de lecturas 32 a 100 pares de bases en una sola ejecución [2]) y con ello la bioinformática ha experimentado en los últimos años una gran evolución; cada vez secuenciar el genoma es más asequible y esto es gracias a las denominadas New Generation Sequencing (NGS) [1], nuevas técnicas más potentes y menos costosas computacionalmente. Dichas técnicas emplean algoritmos de alineamiento que permiten llevar a cabo una secuenciación masiva del ADN [2] con el fin de conocer sus secuencias de nucleótidos y poder compararlas entre individuos de las mismas o diferentes especies, así como hallar el origen evolutivo, la ortología u homología genética... .Existen diversos relativos a la secuenciación genómica que las NGS tratan de abordar, entre ellos la resecuenciación, el mapeo o alineación y el ensamblado "de novo" del genoma[1].

En el presente trabajo nos centramos en el problema de la alineación que trata de, a partir de lecturas de un genoma desconocido, compararlas con un genoma de referencia conocido. Para dicha tarea existen numerosas técnicas que permiten resolver alineamientos pero no todas ellas funcionan de la misma forma. Entre los alineadores más empleados en la actualidad encontramos Bowtie y BWA. Nosotros nos centraremos en este último.

Burrows-Wheeler Aligner (BWA) es un alineador basado en la transformada de Burrows-Wheeler (BWT) que permite, a partir de un genoma de referencia, compararlo con secuencias cortas realizando búsquedas recursivas. Además, a diferencia de otros alineadores como Bowtie, BWA permite errores y gaps (huecos) en las secuencias, que son mutaciones genéticas muy comunes [1]. De esta forma BWA nos ofrece la posibilidad de realizar búsquedas exactas (sin errores ni gaps) o inexactas(con errores y gaps)[3]. En concreto, InexRecur es el algoritmo recursivo empleado por BWA para la búsqueda inexacta. Esta función es de gran utilidad a la hora de alinear secuencias de genoma muy largas, puesto que existe una gran probabilidad de haberse producido errores y gaps. En cambio Bowtie se suele emplear más a menudo en secuencias de menor tamaño en las que se busca una alta correspondencia.

Por otro lado, en cuanto a lenguajes de programación existen un gran número en el mercado. Y en concreto, para el análisis de datos Python y R son los más empleados para ello. En el presente artículo presentaremos el algoritmo en InexRecur en el lenguaje de programación Python y la compararemos con su implementación en R.

2. Métodos

2.1. Precálculo

2.1.1. Árbol de prefijos

El árbol de prefijos es una estructura con forma de árbol obtenida a partir de una cadena de texto de referencia X en la que cada carácter es referenciado por una arista. De esta manera, concatenando los caracteres de una arista de un nodo a la raíz se obtiene un substring único de X o cadena representada por el nodo. Cabe destacar que el árbol de prefijos de X es igual al árbol de sufijos de X invertida (X') Esta herramienta facilita la búsqueda de, dado un substring W, saber si pertenece a X, pues solo habrá que buscar el nodo que representa a W. Podemos observar un ejemplo de un árbol de prefijos en la Fig. 1, 4, 3, 5.

2.1.2. Vector de sufijos (B[i]) para la cadena de referencia X

El primer paso de la búsqueda inexacta en Burrows-Wheeler es la creación del vector de sufijos B[i] dada una cadena de texto de referencia X. B[i] será una cadena de texto con los mismos caracteres que X pero ordenada lexicográficamente [4]. En la Figura 1 mostramos un ejemplo sencillo de la contrucción de B[i] a partir de un X de referencia. Hemos de recalcar que en la práctica, X sería una cadena de texto con un gran número de caracteres (nucleótidos) y W correspondería a una secuencia que queramos alinear con X.

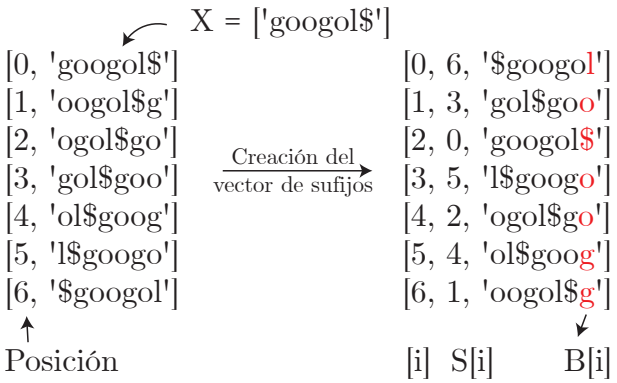

Figura 2: Ejemplo de la creación del vector de sufijos.

2.1.3. Vector C(a)

El vector C(a) en Burrows-Wheeler se obtiene para observar si W es un substring perteneciente a X, y representa el número de caracteres en X[0, n-2] lexicográficamente inferiores a a. En la Figura 2 mostramos un ejemplo del vector C(a).

2.1.4. Vector O(a,i)

Otro vector implicado en BWA es O(a,i) que indica el número de ocurrencias de un símbolo a en B[0,i]. Además, en InexRecur se puede realizar búsquedas inexactas de manera que, dado un W no perteneciente a X pero con un número de errores menores a un umbral z, podemos encontrar secuencias similares en el árbol de prefijos. En la Figura 2 podemos observar un ejemplo de O(a,i).

X = 'googol$'	B[i] = 'lo$oogg'
W = 'lol'	C = (3, 2, 3)

O(a,i)		
i	l	o
0	1	0
1	1	1
2	1	1
3	1	2
4	1	3
5	1	3
6	1	3

Cuadro 1: Obtención de los vectores C(a) y O(a,i) dado W y X.

2.1.5. Vector de sufijos B'[i] para la cadena de referencia invertida X'

Siguiendo con el ejemplo de las Figuras 1 y 2, podríamos obtener a partir de X='googol$' su cadena invertida X='$logoog'. Y el cálculo de B'[i] de X' se realizará de forma análoga al de B[i] para un string X. Como hemos mencionado anteriormente, el árbol de prefijos de X es equivalente al árbol de sufijos de X'.

2.1.6. Vector O'(a,i) para B'[i]

Una vez obtenido B'[i] calculamos su función correspondiente O'(a,i) (Ejemplo en la Cuadro 1). Este es el último paso del precálculo, a partir del cual ya podremos llevar a cabo la búsqueda inexacta.

2.2. Búsqueda inexacta

2.2.1. Función CalculateD

Previamente a comenzar con la función InexRecur deberemos implementar la función CalculateD para obtener el vector D. Este vector hace referencia al número de fallos realizados hasta el momento, a variable z.

El parámetro de entrada a la función será el substring W, y el parámetro de salida será el propio vector D. La lógica llevada a cabo en la función será la mostrada en la Figura 3. El algoritmo establecerá unos valores para el umbral z y para el primer parámetro de W, j, e irá recorriendo en un bucle la subcadena W comprobando si pertenece o no a X. Finalmente asignará el valor de z a D y devolverá este último vector.

```
1  CalculateD(W)
2      z ← 0
3      j ← 0
4      for i = 0 to |W| − 1 do
5      if W[j,i] is not a substring of X then
6          z ← z + 1
7          j ← i + 1
8      D(i) ← z
```

Cuadro 2: Algoritmo de la función CalculateD.[2]

2.2.2. Función de búsqueda inexacta recursiva InexRecur

Esta función es la encargada de realizar la búsqueda inexacta de forma recursiva. Tendrá en cuenta los diferentes tipos posibles de operaciones (match, sustituciones, deleciones e inserciones). Los parámetros de entrada de InexRecur serán el substring W, el número de errores z, y las variables de recursión i, k y l.

```
1  InexRecur(W, i, z, k, l)
2      if z < D(i) then
3          return ∅
4      if i < 0 then
5          return {[k, l]}
6          I ← ∅
7      I ← I ∪ InexRecur(W, i − 1, z − 1, k, l)
8      for each b∈{A,C,G,T} do
9          k ← C(b) + O(b, k − 1) + 1
10         l ← C(b) + O(b, l)
11         if k≤l then
12             I ← I ∪ InexRecur(W, i, z − 1, k, l)
13             if b = W[i] then
14                 I ← I ∪ InexRecur(W, i − 1, z, k, l)
15             else
16                 I ← I ∪ InexRecur(W, i − 1, z − 1, k, l)
17     return I
```

Cuadro 3: Algoritmo de la función InexRecur.[2]

Vemos pues que para cada posible tipo de operaciones el algoritmo llevará a cabo una acción concreta.

En el caso de que se haya producido una deleción (borrado de un carácter o nucleótido en dicha posición) se llamará a la función recursiva disminuyendo una unidad la variable de posición y el número de errores (i, z). Si en cambio se ha observado una inserción (adición de un carácter o nucleótido en la secuencia) se llamará a la función InexRecur y se disminuirá en un tanto el número de errores z.

En el caso de que se haya producido un match (coincidencia) o una mutación de tipo sustitución se volverá a llamar a la función InexRecur pero disminuyendo en una unidad la variable i.

Y por último, si se ha producido una sustitución, se llamará a la función InexRecur disminuyendo en un tanto la variable de posición y el número de errores (i, z), como ocurría en el caso de una deleción.

2.2.3. Función InexSearch

Esta función será la encargada de llamar a la función InexRecur, pasándole sus correspondientes parámetros de entrada W, i, z, k y l. Las variables de entrada a la función InexSearch serán por tanto el substring W y el umbral z. Podemos ver la lógica de esta función en la Figura 5.

```
1  INEXACTSEARCH(W, z)
2          CALCULATED(W)
3          return INEXRECUR(W, |W| − 1, z, 1, |X| − 1)
```

Cuadro 4: Algoritmo de la función INEXSEARCH.[2]

3. Materiales

Los lenguajes que hemos comparado son R, Python y C. La implementación en R es la misma a la usada en las prácticas de la asignatura. Con la implementación proporcionada hemos dividido dos versiones, una que dibuja la traza y otra que calcula el resultado de forma limpia. Todas las versiones del código escrito está disponible en el anejo.

El tiempo y la memoria la hemos medido de dos maneras. Primero hemos evaluado a la función únicamente usando herramientas del lenguaje. En Python hemos usado los paquetes `resource`[5] y `time`[6] para medir el tiempo y la memoria. En R hemos usado la utilidad `system.time`[7] para el tiempo y la libreria `profmem`[8] para perfilar el uso de memoria. En C hemos medido el tiempo usando la librería `time.h`[9] y la memoria usando la herramienta `Valgrind`[10]. Después hemos medido el uso de recursos desde la línea de comandos, llamando a los programas como scripts usando funcionalidades propias de cada lenguaje. Con ello podemos evaluar el sobrecoste de cada una de las tres plataformas usadas. Para ello hemos empleado la herramienta `bench`[11] y el comando `time`[9].

Así pues, vamos a probar el algoritmo de búsqueda recursiva InexRecur con las implementaciones en los lenguajes R, Python y C (pues este último tiene una implementación muy similar a la de Python). Tomaremos como string de referencia X=‘googol$’ y realizaremos varias pruebas con diferentes substrings W=‘gol’; W=‘goog’; W=‘gool’ y asignaremos primero un valor máximo de errores z=0 (equivalente a realizar una búsqueda exacta) y posteriormente un z=1.

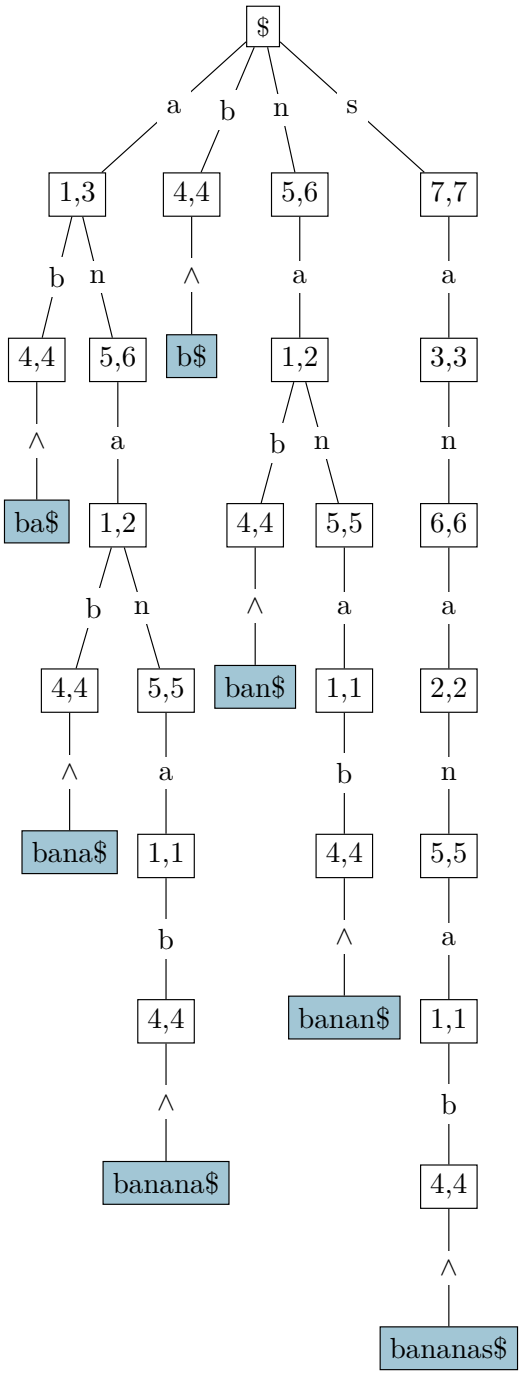

Figura 3: Árbol de prefijos con intervalos SA de la palabra “bananas$”.

4. Resultados

Hemos implementado las trazas de manera que son perfectamente idénticas en Python y en C . A continuación mostramos algunos ejemplos comparativos para verificar que todos los programas se comportan de forma igual. Posteriormente comparamos el uso de tiempo y memoria.

	Mutation	i	z	k	l
Deletion	[l]	1	−1	0	6
Insertion	[g]	2	−1	1	2
Substitution	[g] -> [l]	1	−1	1	2
Insertion	[l]	2	−1	3	3
Match	[l]	1	0	3	3
Deletion	[o]	0	−1	3	3
Insertion	[o]	1	−1	5	5
Match	[o]	0	0	5	5
Deletion	[g]	−1	−1	5	5
Insertion	[g]	0	−1	1	1
Match	[g]	−1	0	1	1
————————	{1,1}	————			
Insertion	[o]	2	−1	4	6
Substitution	[o] -> [l]	1	−1	4	6

Cuadro 5: Traza de InexRecur con X = “googol$”, W = “gol”, z = 0 en C y Python

```
|      INEXRECUR   -    by XAVI GABRI AITANA ALFREDO      |
-     D       [ l ]         1     -1     0     6
-     I       [ g ]         2     -1     1     2
-     S    [ g -> l ]    1     -1     1     2
-     I       [ l ]         2     -1     3     3
-     M       [ l ]         1     0     3     3
-     -     D       [ o ]         0     -1     3     3
-     -     I       [ o ]         1     -1     5     5
-     -     M       [ o ]         0     0     5     5
-     -     -     D       [ g ]         -1     -1     5     5
-     -     -     I       [ g ]         0     -1     1     1
-     -     -     M       [ g ]         -1     0     1     1
?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?->   [ c(1, 1) ]
-     I       [ o ]         2     -1     4     6
-     S    [ o -> l ]    1     -1     4     6
```

Cuadro 6: Traza de InexRecur con X = “googol$”, W = “gol”, z = 0 en R

```
                              Mutation    i   z   k   l
 Deletion                       [g]       2  −1   0   6
 Insertion                      [g]       3  −1   1   2
 Match                          [g]       2   0   1   2
  Deletion                      [o]       1  −1   1   2
  Insertion                     [o]       2  −1   4   4
  Match                         [o]       1   0   4   4
   Deletion                     [o]       0  −1   4   4
   Insertion                    [o]       1  −1   6   6
   Match                        [o]       0   0   6   6
 Insertion                      [l]       3  −1   3   3
 Substitution              [l] −> [g]     2  −1   3   3
 Insertion                      [o]       3  −1   4   6
 Substitution              [o] −> [g]     2  −1   4   6
```

Cuadro 7: Traza de InexRecur con X = “googol$”, W = “goog”, z = 0 en C y Python

```
|      INEXRECUR   −   by XAVI GABRI AITANA ALFREDO      |
−    D       [ g ]      2    −1    0    6
−    I       [ g ]      3    −1    1    2
−    M       [ g ]      2     0    1    2
−    −    D       [ o ]      1    −1    1    2
−    −    I       [ o ]      2    −1    4    4
−    −    M       [ o ]      1     0    4    4
−    −    −    D       [ o ]      0    −1    4    4
−    −    −    I       [ o ]      1    −1    6    6
−    −    −    M       [ o ]      0     0    6    6
−    I       [ l ]      3    −1    3    3
−    S    [ l −> g ]    2    −1    3    3
−    I       [ o ]      3    −1    4    6
−    S    [ o −> g ]    2    −1    4    6
```

Cuadro 8: Traza de InexRecur con X = “googol$”, W = “goog”, z = 0 en R

1	Mutation	i	z	k	l
2	Deletion [l]	2	0	1	6
3	Deletion [o]	1	−1	1	6
4	Insertion [g]	2	−1	1	2
5	Substitution [g] −> [o]	1	−1	1	2
6	Insertion [o]	2	−1	4	6
7	Match [o]	1	0	4	6
8	Deletion [o]	0	−1	4	6
9	Insertion [g]	1	−1	1	2
10	Substitution [g] −> [o]	0	−1	1	2
11	Insertion [o]	1	−1	6	6
12	Match [o]	0	0	6	6
13	Deletion [g]	−1	−1	6	6
14	Insertion [g]	0	−1	2	2
15	Match [g]	−1	0	2	2
16	{2,2}				
17	Insertion [g]	3	0	1	2
18	Substitution [g] −> [l]	2	0	1	2
19	Deletion [o]	1	−1	1	2
20	Insertion [o]	2	−1	4	4
21	Match [o]	1	0	4	4
22	Deletion [o]	0	−1	4	4
23	Insertion [o]	1	−1	6	6
24	Match [o]	0	0	6	6
25	Deletion [g]	−1	−1	6	6
26	Insertion [g]	0	−1	2	2
27	Match [g]	−1	0	2	2
28	{2,2}				
29	Insertion [o]	3	0	4	6
30	Substitution [o] −> [l]	2	0	4	6
31	Deletion [o]	1	−1	4	6
32	Insertion [g]	2	−1	1	2
33	Substitution [g] −> [o]	1	−1	1	2
34	Insertion [o]	2	−1	6	6
35	Match [o]	1	0	6	6
36	Deletion [o]	0	−1	6	6
37	Insertion [g]	1	−1	2	2
38	Substitution [g] −> [o]	0	−1	2	2

Cuadro 9: Traza de InexRecur con X = “googol$”, W = “gool”, z = 1 en C y Python

```
1  |      INEXRECUR  -  by XAVI GABRI AITANA ALFREDO      |
2  -    D       [ l ]        2    0    1    6
3  -    -    D       [ o ]        1   -1    1    6
4  -    -    I       [ g ]        2   -1    1    2
5  -    -    S    [ g -> o ]     1   -1    1    2
6  -    -    I       [ o ]        2   -1    4    6
7  -    -    M       [ o ]        1    0    4    6
8  -    -    -    D       [ o ]        0   -1    4    6
9  -    -    -    I       [ g ]        1   -1    1    2
10 -    -    -    S    [ g -> o ]     0   -1    1    2
11 -    -    -    I       [ o ]        1   -1    6    6
12 -    -    -    M       [ o ]        0    0    6    6
13 -    -    -    -    D       [ g ]        -1   -1    6    6
14 -    -    -    -    I       [ g ]        0   -1    2    2
15 -    -    -    -    M       [ g ]        -1    0    2    2
16 ?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-> [ c(2, 2) ]
17 -    I       [ g ]        3    0    1    2
18 -    S    [ g -> l ]     2    0    1    2
19 -    -    D       [ o ]        1   -1    1    2
20 -    -    I       [ o ]        2   -1    4    4
21 -    -    M       [ o ]        1    0    4    4
22 -    -    -    D       [ o ]        0   -1    4    4
23 -    -    -    I       [ o ]        1   -1    6    6
24 -    -    -    M       [ o ]        0    0    6    6
25 -    -    -    -    D       [ g ]        -1   -1    6    6
26 -    -    -    -    I       [ g ]        0   -1    2    2
27 -    -    -    -    M       [ g ]        -1    0    2    2
28 ?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-?-> [ c(2, 2) ]
29 -    I       [ o ]        3    0    4    6
30 -    S    [ o -> l ]     2    0    4    6
31 -    -    D       [ o ]        1   -1    4    6
32 -    -    I       [ g ]        2   -1    1    2
33 -    -    S    [ g -> o ]     1   -1    1    2
34 -    -    I       [ o ]        2   -1    6    6
35 -    -    M       [ o ]        1    0    6    6
36 -    -    -    D       [ o ]        0   -1    6    6
37 -    -    -    I       [ g ]        1   -1    2    2
38 -    -    -    S    [ g -> o ]     0   -1    2    2
```

Cuadro 10: Traza de InexRecur con X = "googol$", W = "gool", z = 1 en R

Se puede comprobar que las tres trazas contienen la misma información. Procedemos ahora a la comparación de los tiempos medidos, primero las mediciones tomadas desde el propio programa.

```
inexrecur_time.c
12.34µs from 10000 iterations.

inexrecur_time.py
real     sys     user
268.89µs 0.37µs 268.16µs

inexrecur_time.R
user      system    elapsed
5422µs    18µs      5443µs
```

Cuadro 11: Tiempos de "CPU".

En cada caso tomamos múltiples lecturas y hacemos la media. Como es de esperar la solución compilada de C es ∼20 veces más rápida al script de Python y R parece ser ∼450 veces más lento que C y ∼20 veces más lento que Python . Para comparar el efecto del interpretador medimos ahora el tiempo ejecutando desde la línea de comandos. LLamamos a los scripts usando `#!/bin/env Rscript` y `#!/bin/env Python`.

```
bench ./inexrecur_clean ./inexrecur_clean.py ./inexrecur_clean.R
benchmarking bench/./inexrecur_clean
time                 4.524 ms    (4.496 ms .. 4.551 ms)
                     0.999 R²    (0.999 R² .. 1.000 R²)
mean                 4.522 ms    (4.499 ms .. 4.559 ms)
std dev              89.83 µs    (58.07 µs .. 133.1 µs)

benchmarking bench/./inexrecur_clean.py
time                 39.26 ms    (39.12 ms .. 39.40 ms)
                     1.000 R²    (1.000 R² .. 1.000 R²)
mean                 39.30 ms    (39.18 ms .. 39.45 ms)
std dev              266.5 µs    (177.4 µs .. 388.7 µs)

benchmarking bench/./inexrecur_clean.R
time                 278.5 ms    (273.6 ms .. 285.0 ms)
                     1.000 R²    (1.000 R² .. 1.000 R²)
mean                 280.6 ms    (279.1 ms .. 281.9 ms)
std dev              1.714 ms    (1.027 ms .. 2.517 ms)
variance introduced by outliers: 16% (moderately inflated)
```

Cuadro 12: Tiempo de "pared" según la utilidad *bench*.

En este caso C es ∼10 veces más rápido a Python y ∼70 veces más rápido que R , que continua siendo ∼7 veces más lento que Python .

```
inexrecur_clean.c
==81469==      in use at exit: 18,588 bytes in 164 blocks
==81469==    total heap usage: 191 allocs, 27 frees, 24,894 bytes allocated

inexrecur_mem.py
14,007,812 bytes

inexrecur_mem.R
4,932,584 bytes
```

Cuadro 13: Memoria usada medido desde el script.

La memoria usada es bastante mayor en Python que en R cuando la medimos sin tener en cuenta el entorno de ejecución, C utiliza sustancialmente menos memoria.

```
time ./inexrecur_clean
(5,5)
(1,1)
./inexrecur_clean
0.00s  user 0.00s system 44% cpu 0.004 total
max memory:                 1676 kB

time ./inexrecur_clean.py
(1, 1)
(5, 5)
./inexrecur_clean.py
0.03s  user 0.01s system 82% cpu 0.039 total
max memory:                 6272 kB

time ./inexrecur_clean.R
[[1]]
[1] 5 5

[[2]]
[1] 1 1

./inexrecur_clean.R
0.23s  user 0.04s system 97% cpu 0.277 total
max memory:                 67476 kB
```

Cuadro 14: Memoria usada medido desde fuera del script.

Cuando consideramos el entorno de ejecución interpretado al completo C se mantiene a la cabeza con la sorpresa de que R emplea ∼10 veces más memoria que Python en este ejemplo en particular.

5. Discusión

Como se ha visto en el apartado de Resultados, hemos implementado el algoritmo BWA en los lenguajes R, Python y C; hemos utilizado distintos criterios para evaluar el rendimiento de éstos y realizar una comparación de su funcionamiento. Dichos criterios son el tiempo de procesamiento y la memoria usada. En el caso del tiempo, centrándonos en los dos lenguajes que queremos comparar en este artículo, el que ofrece un tiempo menor medido desde el propio programa a la hora de alinear las secuencias es Python, mientras que R resulta ser más lento en comparación con éste. Por otro lado, evaluando la memoria, Python ocupa aproximadamente 10 veces menos memoria que R midiendo desde fuera del script.

Python es un lenguaje con mensaje interactivo, permite la escritura dinámica y administra la memoria automáticamente. Tiene una sintaxis muy clara e intuitiva, y utiliza márgenes para hacer los bloques de declaraciones. Además, se trata de un software libre por lo que se puede acceder a él fácilmente y desde todos los sistemas operativos. [12]

R se trata de un lenguaje de programación con enfoque de análisis estadístico, no se suele usar para implementar programas largos como se hace con otros lenguajes. Permite combinar programación imperativa con funcional, además de incluir programación orientada a objetos. Asimismo, es uno de los lenguajes más utilizados en investigación científica. [13, 14]

Así pues, basándonos en los ejemplos que hemos ejecutado, parece ser que en Python es más sencillo implementar el algoritmo InexRecur (ver Anexo) y además presenta un mejor valor en los criterios evaluados anteriormente. Por lo tanto, dado que alinear secuencias es un procedimiento que ya de por sí consume mucho tiempo, podríamos decir que a la hora de mapear una secuencia corta con un genoma de referencia sería más rápido y eficaz utilizar el lenguaje Python [15]. Sin embargo, hay que tener en cuenta que como este algoritmo es computacionalmente costoso no será conveniente utilizarlo para alinear secuencias cortas con, por ejemplo, el genoma humano completo, ya que daría unos tiempos de procedimiento demasiado elevados [16]. De la misma forma, también se comenta este mismo inconveniente en el artículo de Heng Li et al., 2009 [2] donde se expone que el rendimiento del alineador BWA se reduce cuando son lecturas largas, por lo que aconsejan que a la hora de trabajar con secuencias largas se divida la lectura en varios fragmentos cortos, alinearlos por separado y posteriormente unir las alineaciones parciales en una global.

6. Conclusiones

Hemos propuesto una implementación del algoritmo InexRecur en Python para alineamientos inexactos de fragmentos de secuencias con secuencias largas. Lo implementamos de forma correcta y comprobamos su funcionamiento con varios ejemplos con la traza que se nos ha proporcionado. Se comprueba que la programación de este algoritmo en Python proporciona un gran potencial para disminuir el tiempo de procesamiento y la memoria total utilizada, además de ofrecer una sintaxis más clara en comparación con la implementación en R.

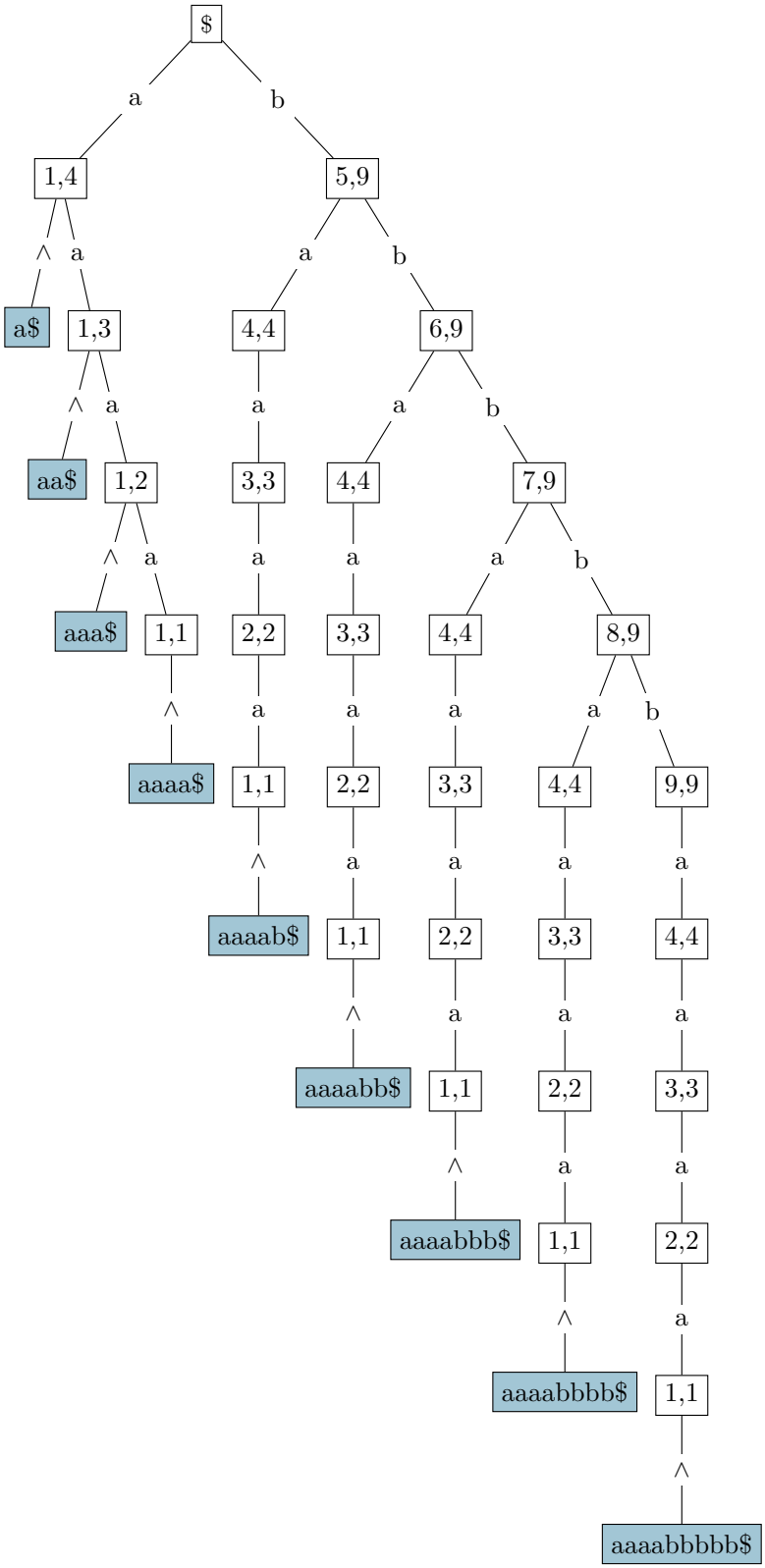

Figura 4: Árbol de prefijos con intervalos SA de la palabra “aaaabbbbb$”.

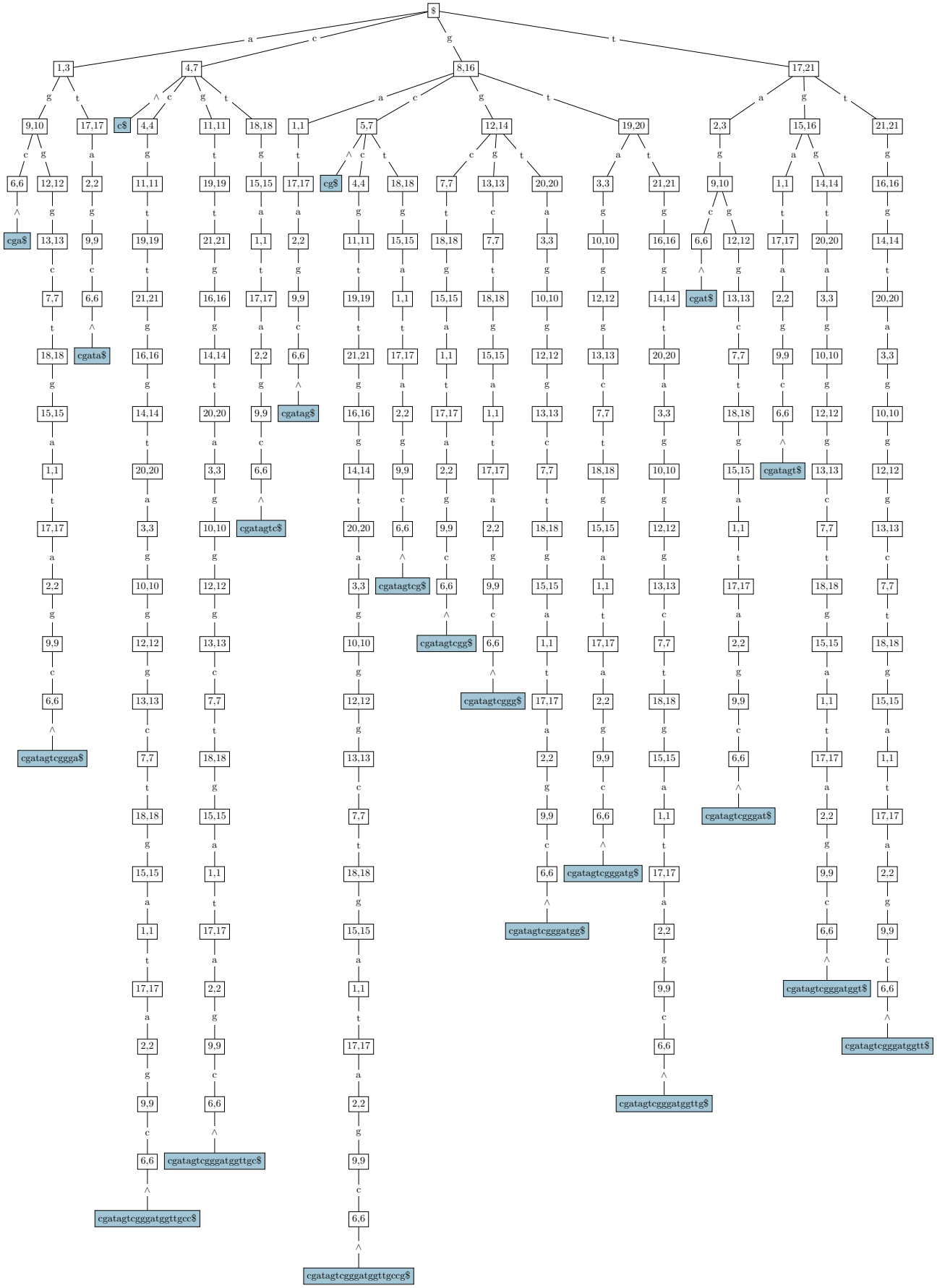

Figura 5: Árbol de prefijos con intervalos SA de la palabra “cgatagtcgggatggttgccg$”.

7. Anejo

Listing 1: Python inexrecur_clean.py

```
#!/usr/bin/env python
# Started Wed 25 Mar 00:56:18 2020 by nahnero. #
def CalculateD (W):
    D = [0]*len (W)
    z, j = 0, 0
    for i in range (len (W)):
        if W[j:i] not in X:
            z += 1
            j = i + 1
        D[i] = z
    return D

def InexRecur (W, i, z, k, l):
    if i < 0:
        if z < 0:
            return set ([])
        else:
            return set ([(k, l)])
    if z < D[i]:
        return set ([])
    I = set ([])
    I = I.union (InexRecur (W, i-1, z-1, k, l))
    for bi in dic:
        ki = C (bi) + O (bi, k-1) + 1
        li = C (bi) + O (bi, l)
        if ki <= li:
            I = set (I.union (InexRecur (W, i, z-1, ki, li)))
            if bi == W[i]:
                I = set (I.union (InexRecur (W, i-1, z, ki, li)))
            else:
                I = set (I.union (InexRecur (W, i-1, z-1, ki, li)))
    return I

def InexSearch (W, z):
    return InexRecur (
            W = W,
            i = len (W)-1,
            z = z,
            k = 0,
            l = len (X)-1)
X = 'googol$';
W = 'lol'
dic = ''.join (sorted (set (X.replace ('$', ''))))

BWT  = [[(X*2)[i:i+len (X)], i] for i in range (len (X))]
BWT.sort ()
S, B = [], ''
for i in BWT: S.append (i[1]); B += (i[0][-1])
C = lambda a: ''.join (sorted (X.replace ('$', '')))[0:-2].find (a)
O = lambda a, i: B[0:i+1].count (a)

D  = CalculateD (W)
IS = InexSearch (W, 1)

for i in IS: print (i)
```

Listing 2: R inexrecur_clean.R

```
#!/usr/bin/env Rscript
C <- function(a) {
  components <- gsub("[[:punct:]]","",X)
  components <- strsplit(components,"")
  components_sorted <- sort(components[[1]])
  idx <- min(which(components_sorted == a))
  C_num <- idx-1
  return(C_num)
}
O <- function(a,i) {
  i<-i+1
  coincidences <- grep(a,B)
  counter <- i>=coincidences
  O_num <- length(counter[counter==TRUE])
  return(O_num)
}

INEXRECUR <- function(W,i,z,k,l,sep) {
  if (i<0){
    if (z<0) {return({})}
    else {
      result <- list(c(k,l))
      return(result)
    }}
  if (z<D[i+1]) {return({})}
  I <- {}
  I <- c (I, INEXRECUR(W,i-1,z-1,k,l,sep));
  for (bi in b){
    ki <- C(bi) + O(bi,k-1) + 1  #Precalculate C and O function
    li <- C(bi) + O(bi,l)  #Precalculate C and O function
    if (ki<=li){
      I <- c (I, INEXRECUR(W,i,z-1,ki,li,sep));
      if (bi==W[[1]][i+1]){
        I <- c (I, INEXRECUR(W,i-1,z,ki,li,sep))
      }else{
        I <- c (I, INEXRECUR(W,i-1,z-1,ki,li,sep))
      }}}
  return(I)

} #end INEXRECUR

X <- "googol$"
SA_pre <- c()
SA_pre[1] <- X
for (i in 2:nchar(X)){
SA_pre[i] <- paste(substring(X,i,nchar(X)),
                   substring(X,1,i-1),sep='',collapse=NULL)
}
SA_list<-sort(SA_pre,index.return=TRUE)
SA<-SA_list$x
S<-SA_list$ix
B<-paste(substring(SA,nchar(X),nchar(X)),sep='',collapse=NULL)
b <- c("g","l","o")
D <- c(0,0,1)
W = list(c("l","o","l"))
i = 2
z = 1
a = INEXRECUR(W,i,z,0,6,"")
print (a)
```

Listing 3: C inexrecur_clean.c

```
int main (void){
        char X[] = "googol$";
        char W[] = "lol";
        char dict[] = "glo";
        setup (X, W, dict);
        return 0;
}


void setup (char* X1, char* W1, char* dic1){

        int i, length; i = length = 0;
        X     = X1;
        /* reverse */
        W     = W1;
        while (W1[++i]);
        Wlen  = i;
        dic   = dic1;

        while (X[++length]);

        D   = (int*)  calloc (length, sizeof (int));
        BWT = (char**)calloc (length, sizeof (char*));
        Xs  = (char*) calloc (length, sizeof (char));
        B   = (char*) calloc (length, sizeof (char));
        X2  = (char*) calloc (2*length+1, sizeof (char));
        for (i = 0; i != length; i++)
                BWT[i] = (char*)malloc (length*sizeof (char)+1);

        sprintf (X2, "%s%s", X, X);

        for (i = 0; i != length; i++)
                sprintf (BWT[i], "%.*s", (int)length, X2+i);


        sort (BWT, length);
        CalculateD (W1);

        for (i = 0; i != length; i++) B [i] = BWT[i][length-1];
        for (i = 0; i != length; i++) Xs[i] = BWT[i][0];

        i = 0; while (X[++i]);
        InexRecur (W1, Wlen - 1, 1, 0, i - 1);

        free (Xs);
        free (X2);
        free (D);
        for (i = 0; i != length ; i++) free (BWT[i]);
        free (BWT);
}

static int myCompare(const void* a, const void* b){
        return strcmp (*(const char**)a, *(const char**)b);
}

void sort(char* arr[], int n){
        qsort (arr, n, sizeof(char*), myCompare);
}
```

Listing 4: C inexrecur_clean.c (cont.)

```
59
60  int C (char a){
61          int i = 0;
62          while (X[++i]) if (Xs[i] == a) return i - 1;
63          return 0;
64  }
65
66  int O (char a, int i){
67          int acc = 0, j;
68          for (j = 0; j <= i; j++) {
69                  if (B[j] == a) acc++;
70          }
71          return acc;
72  }
73
74  /* Ignacio Amat */
75
76  void CalculateD (char* W){
77          short z = 0, j = 0, i;
78          char buf[BUFSIZ] = {'\0'};
79          extern int Wlen;
80          for (i = 0; i != Wlen; i++){
81                  sprintf (buf, "%.*s", i, W+j);
82                  if (!strstr (X, buf)){
83                          z++;
84                          j = i + 1;
85                  }
86                  D[i] = z;
87          }
88  }
89
90  void InexRecur (char* W, int i, int z, int k, int l){
91          if (i < 0){
92                  if (z < 0){
93                          return;
94                  } else {
95                          printf ("(%d,%d)\n", k, l);
96                          return;
97                  }
98          }
99          if (z < D[i]){return;}
100         InexRecur (W, i-1, z-1, k, l);
101         int ki, li, m;
102         for (m = 0; m != 3; m++){
103                 ki = C (dic [m]) + O (dic [m], k-1) + 1;
104                 li = C (dic [m]) + O (dic [m], l);
105                 if (ki <= li){
106                         InexRecur (W, i, z-1, ki, li);
107                         if (dic [m] == W[i]){
108                                 InexRecur (W, i-1, z, ki, li);
109                         } else {
110                                 InexRecur (W, i-1, z-1, ki, li);
111                         }
112                 }
113         }
114         return;
115 }
```

Referencias

[1] D. Giménez, "Implementación y análisis de algoritmos de alineación para datos de next generation sequencing (ngs)," Ph.D. dissertation, Universidad Autónoma de Madrid (UAM), 2016.

[2] H. Li and R. Durbin, *Fast and accurate short read alignment with Burrows-Wheeler transform.*, 2009. [Online]. Available: https://doi.org/10.1093/bioinformatics/btp324

[3] J. M. García Gómez, *Bioinformatics Burrows-Wheeler Alignment (BWA).* Apuntes de la asignatura de Bioinformática, Grado en Ingeniería Biomédica, UPV, 2020. [Online]. Available: https://poliformat.upv.es/access/content/group/GRA_13024_2019/JuanMGarcia-Gomez/Semana%204/BWA.pdf

[4] J. G. García Pardo, A. Torregrosa Lloret, J. De la Torre Costa, and A. Pascual Belda, *InexRecur.* Apuntes de la asignatura de Bioinformática, Grado en Ingeniería Biomédica, UPV, 2020. [Online]. Available: https://poliformat.upv.es/access/content/group/GRA_13024_2019/JuanMGarcia-Gomez/Semana%204/INEXRECUR.pdf

[5] "resource - resource usage information." [Online]. Available: https://docs.python.org/3/library/resource.html

[6] "time - time access and conversions." [Online]. Available: https://docs.python.org/3/library/time.html?highlight=time#module-time

[7] "system.time." [Online]. Available: https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/system.time

[8] H. Bengtsson, "Simple memory profiling for r [r package profmem version 0.5.0]." [Online]. Available: https://cran.r-project.org/web/packages/profmem/index.html

[9] "time.h," https://pubs.opengroup.org/onlinepubs/007908799/xsh/time.h.html, (Accessed on 05/02/2020).

[10] "Valgrind home," https://valgrind.org/, (Accessed on 05/02/2020).

[11] "Github - gabriel439/bench: Command-line benchmark tool," https://github.com/Gabriel439/bench, (Accessed on 05/02/2020).

[12] H. Fangohr, *A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering*, 2004. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F978-3-540-25944-2_157.pdf

[13] C. J. Gil Bellosta, *R para profesionales de los datos: una introducción*, 2018. [Online]. Available: https://www.datanalytics.com/libro_r/programacion-en-r.html

[14] C. Kleiber and A. Zeileis, *Applied Econometrics with R*, 1st ed. Springer Science+Business Media, 2008.

[15] J. M. Abuín, J. C. Pichel, T. F. Pena, and J. Amigo, *BigBWA: approaching the Burrows–Wheeler aligner to Big Data technologies.* International Society for Computational Biology, 2015. [Online]. Available: https://doi.org/10.1093/bioinformatics/btv506

[16] E. J. Houtgast, V.-M. Sima, K. Bertels, and Z. Al-Ars, "An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm," in *2015 International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS)*. IEEE, Jul. 2015. [Online]. Available: https://doi.org/10.1109/samos.2015.7363679