Trabajo Bioinformática*

Ignacio Amat Hernández Ángela LLopis Hernández Estrella Ballester Hoyo

1 de mayo de 2020

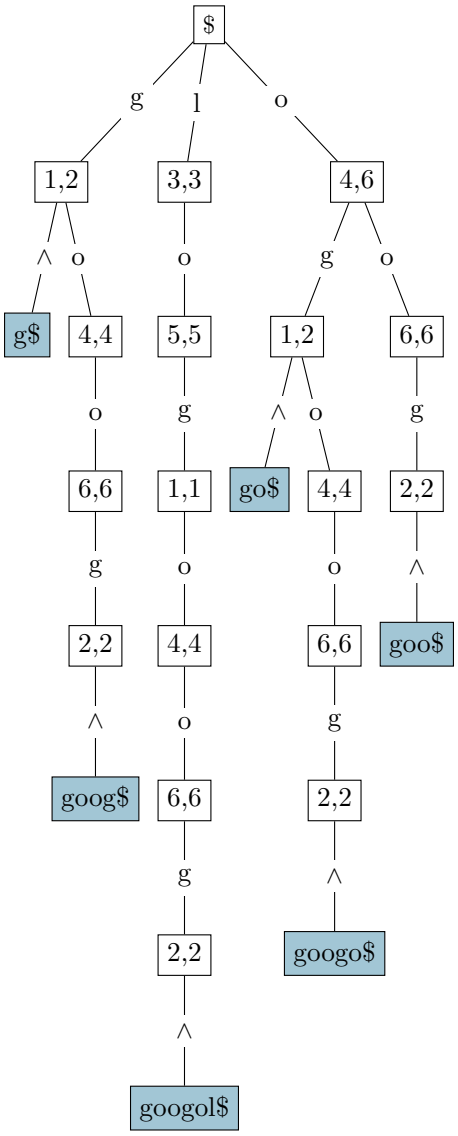

Figura 1: Árbol de prefijos con intervalos SA de la palabra “googol$”.

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

1. Materiales

Los lenguajes que hemos comparado son , python y . La implementación en 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` y `time` para medir el tiempo y la memoria. En hemos usado la utilidad `system.time` para el tiempo y la libreria `profmem` para perfilar el uso de memoria. En hemos medido el tiempo usando la librería `time.h` y la memoria usando la herramienta `Valgrind`. 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` y el comando `time`.

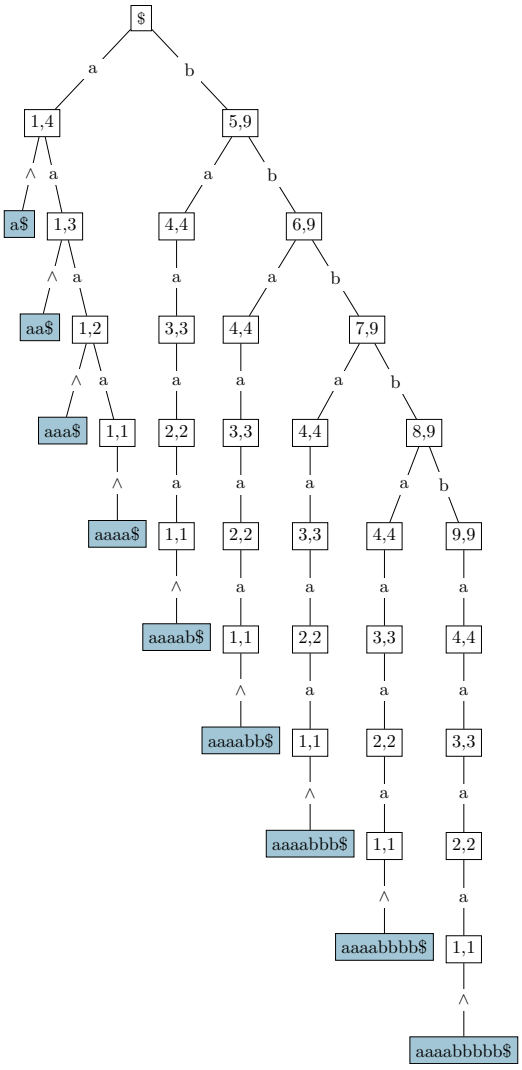

Figura 2: Árbol de prefijos con intervalos SA de la palabra “aaaabbbbb$”.

2. 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 1: 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 2: 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 3: 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 4: Traza de INEXRECUR con X = “googol$”, W = “goog”, z = 0 en R

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

Cuadro 5: Traza de InexRecur con X = “googol$”, W = “gool”, z = 1 en  y 

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

Cuadro 6: Traza de InexRecur con X = “googol$”, W = “gool”, z = 1 en 

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 7: Tiempos de "CPU".

En cada caso tomamos múltiples lecturas y hacemos la media. Como es de esperar la solución compilada de es ∼20 veces más rápida al script de y parece ser ∼450 veces más lento que y ∼20 veces más lento que . 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 8: Tiempo de "pared" según la utilidad *bench*.

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

```
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
6,631,424 bytes

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

Cuadro 9: Memoria usada medido desde el script.

La memoria usada es comparable en R y python cuando la medimos sin tener en cuenta el entorno de ejecución, C utiliza entorno a 200 veces menos memoria.

```
time ./inexrecur_clean
(5,5)
(1,1)
./inexrecur_clean
0.00s  user 0.00s system 44% cpu 0.004 total
max memory:                 676 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 10: 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.

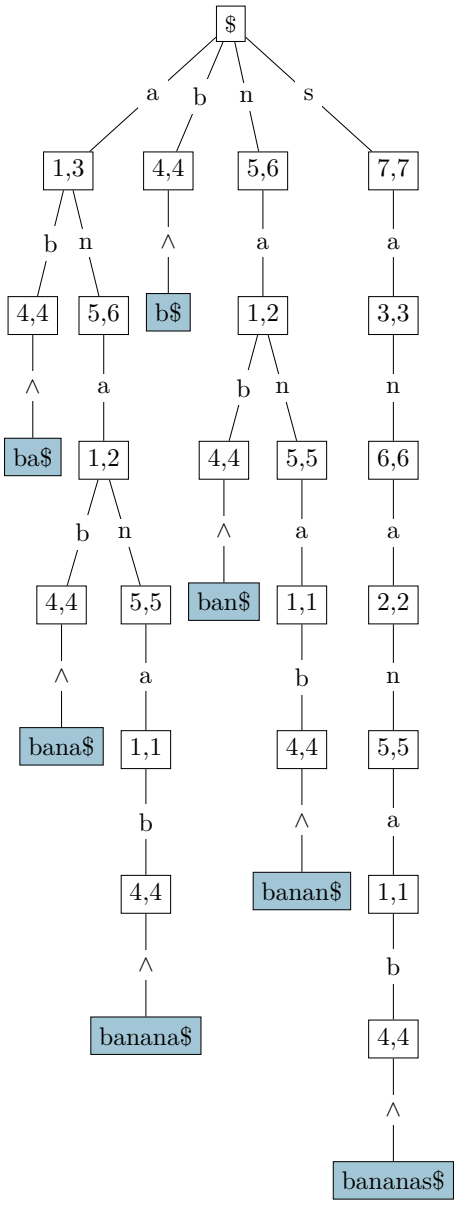

Figura 3: Árbol de prefijos con intervalos SA de la palabra “bananas$”.

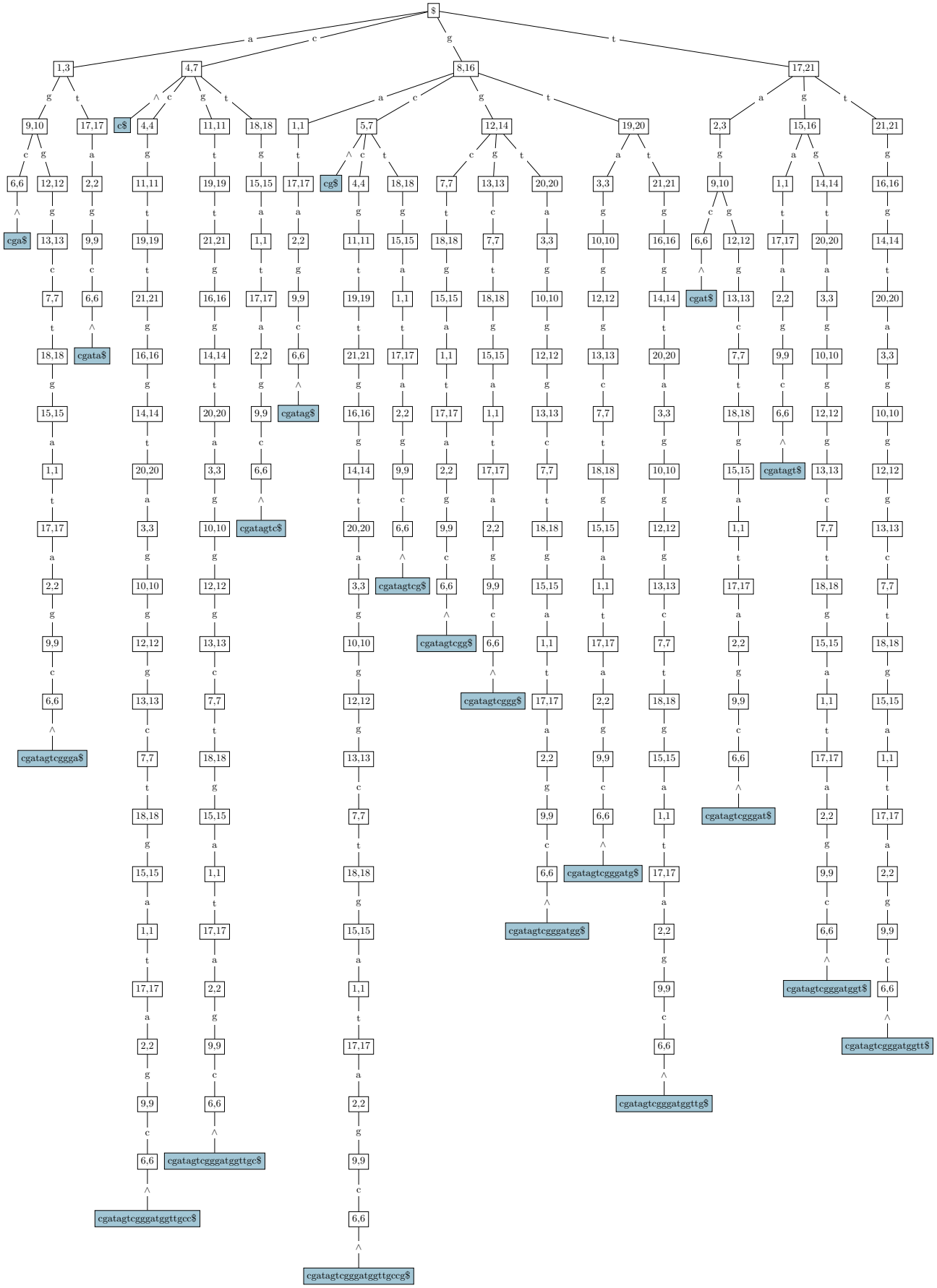

Figura 4: Árbol de prefijos con intervalos SA de la palabra “cgatagtcgggatggttgccg$”.

3. 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: 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: inexrecur_clean.c

```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);
}
```

```
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 }
```