

# Resolución de Tarea 7 - Cadenas de Caracteres y Geometría Computacional (Fecha: 17 de Noviembre de 2025)

Universidad Simón Bolívar  
Departamento de Computación y Tecnología de la Información  
CI5651 - Diseño de Algoritmos I  
Septiembre - Diciembre 2025  
Estudiante: Junior Miguel Lara Torres (17-10303)

## Tarea 7 (9 puntos)

### Indice

- Resolución de Tarea 7 - Cadenas de Caracteres y Geometría Computacional (Fecha: 17 de Noviembre de 2025)
- Indice
- Pregunta 1
  - (a) Construcción del Árbol de Sufijos para  $w$
  - (b) Construcción del Arreglo de Sufijos (SA)
  - (c) Cálculo de PLCP[k] y LCP[k]
    - \* 1. Cálculo de Phi
    - \* 2. Cálculo de PLCP[k] (Prefijo Común Permutado más Largo)
    - \* 3. Cálculo de LCP[i] (Prefijo Común más Largo)
- Pregunta 2
  - Implementación en C++

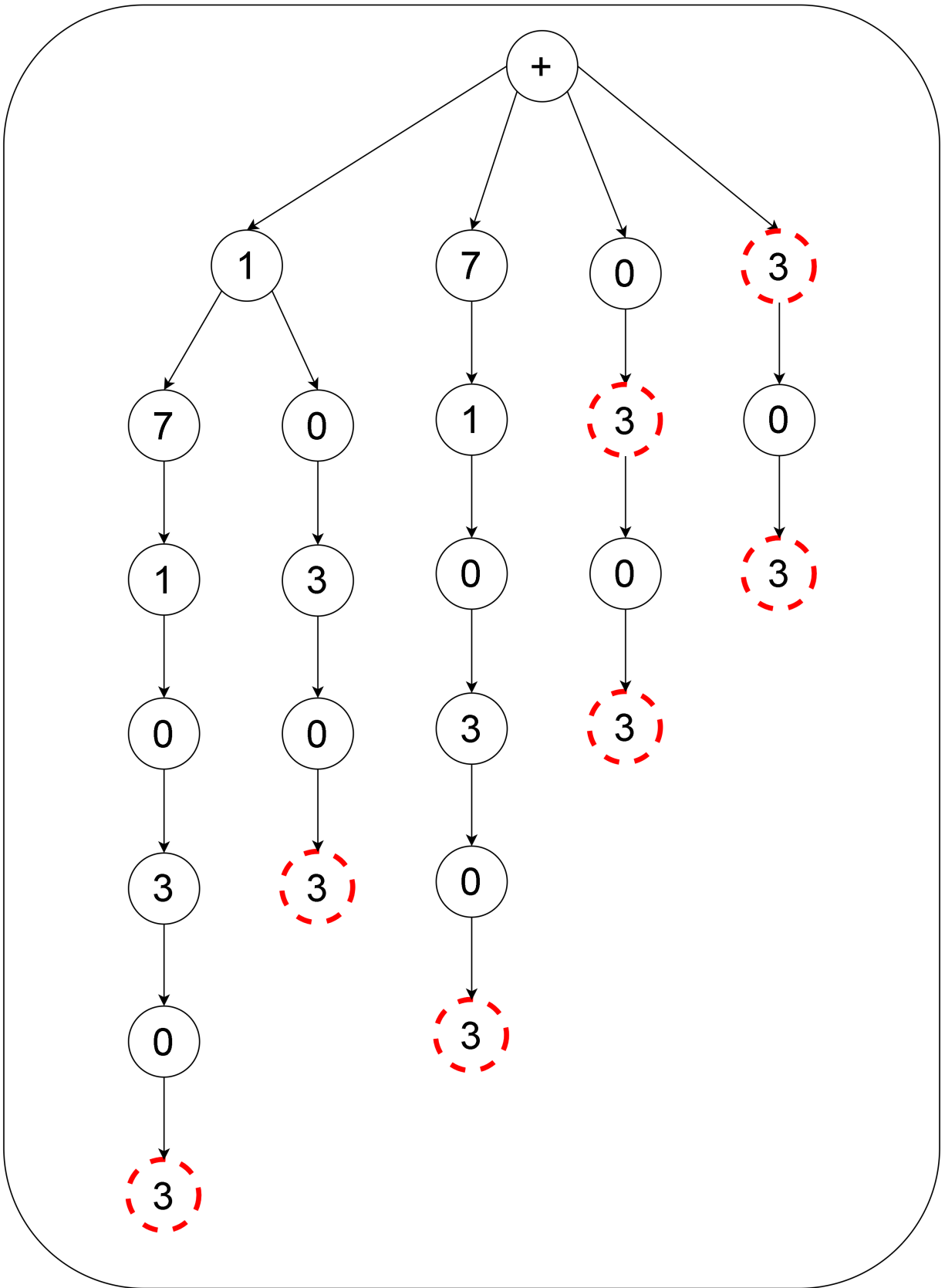
### Pregunta 1

El carné a utilizar es “17-10303”. La cadena  $n$  a considerar, sin el guión, es  $w = "1710303"$ . La longitud de la cadena es  $N = 7$ .

#### (a) Construcción del Árbol de Sufijos para $w$

Un árbol de sufijos se construye a partir de la cadena  $w$  al organizar todos sus sufijos en una estructura de árbol que aprovecha los prefijos compartidos, similar a un Trie. En este caso, la estructura contendría todos los sufijos de  $w = "1710303"$ :

1. 1710303
2. 710303
3. 10303
4. 0303
5. 303
6. 03
7. 3



## (b) Construcción del Arreglo de Sufijos (SA)

El arreglo de sufijos ( $SA$ ) almacena los índices de inicio de los sufijos de  $w$  en el orden en que aparecerían si estuvieran ordenados lexicográficamente.

Los sufijos ordenados de  $w = "1710303"$  son:

Índice	Sufijo
7	$\lambda$
3	0303
5	03
2	10303
0	1710303
4	303
6	3
1	710303

El Arreglo de Sufijos ( $SA$ ) es:

$$SA = [7, 3, 5, 2, 0, 4, 6, 1]$$

## (c) Cálculo de PLCP[k] y LCP[k]

Para calcular los arreglos de Longitud de Prefijo Común (LCP) y Prefijo Común Permutado más Largo (PLCP), primero se define el arreglo auxiliar  $\Phi$  (Phi).

### 1. Cálculo de Phi

El arreglo  $\Phi$  almacena, para el sufijo que empieza en la posición  $k$ , el índice de inicio del sufijo que lo precede inmediatamente en el arreglo de sufijos ordenado ( $SA$ ).

$$\Phi[SA[i]] = SA[i - 1]$$

El caso base es  $SA = 7$  (el sufijo  $\lambda$ ), para el cual  $\Phi$  se define como  $\lambda$  o  $-1$ .

Rank (i)	$SA[i]$ (Índice)	Sufijo $w[SA[i]..]$	$SA[i - 1]$ (Predecesor)	$[SA[i]]$
0	7	$\lambda$	N/A	-1
1	3	0303	7	7
2	5	03	3	3
3	2	10303	5	5
4	0	1710303	2	2
5	4	303	0	0
6	6	3	4	4
7	1	710303	6	6

Arreglo  $\Phi$  (Indexado por posición  $k$  de 0 a 7):

$$\Phi = [2, 6, 5, 7, 0, 3, 4, -1]$$

## 2. Cálculo de PLCP[k] (Prefijo Común Permutado más Largo)

$PLCP[k]$  es la longitud del prefijo común más largo entre el sufijo que comienza en  $k$  ( $w[k..]$ ) y el sufijo que comienza en  $\Phi[k]$  ( $w[\Phi[k]..]$ ).

k	$\Phi[k]$	Sufijo $w[k..]$	Sufijo $w[\Phi[k]..]$	<b>PLCP[k]</b>
0	2	<b>1710303</b>	<b>10303</b>	<b>1</b>
1	6	<b>710303</b>	<b>3</b>	<b>0</b>
2	5	<b>10303</b>	<b>03</b>	<b>0</b>
3	7	<b>0303</b>	<b><math>\lambda</math></b>	<b>0</b>
4	0	<b>303</b>	<b>1710303</b>	<b>0</b>
5	3	<b>03</b>	<b>0303</b>	<b>2</b>
6	4	<b>3</b>	<b>303</b>	<b>1</b>
7	-1	<b><math>\lambda</math></b>	<b>N/A</b>	<b>0</b>

**Arreglo PLCP** (Indexado por posición  $k$  de 0 a 7):

$$PLCP = [1, 0, 0, 0, 0, 2, 1, 0]$$

## 3. Cálculo de LCP[i] (Prefijo Común más Largo)

$LCP[i]$  es la longitud del prefijo común más largo entre el sufijo de rango  $i$  ( $SA[i]$ ) y el sufijo de rango  $i - 1$  ( $SA[i - 1]$ ). Se calcula utilizando el arreglo PLCP mediante la fórmula  $LCP[i] = PLCP[SA[i]]$ .

$LCP$  se define como 0.

Rank (i)	$SA[i]$	Sufijo $w[SA[i]..]$	Sufijo $w[SA[i - 1]..]$	$PLCP[SA[i]]$	<b>LCP[i]</b>
0	7	<b><math>\lambda</math></b>	<b>N/A</b>	<b>N/A</b>	<b>0</b>
1	3	<b>0303</b>	<b><math>\lambda</math></b>	<b><math>PLCP = 0</math></b>	<b>0</b>
2	5	<b>03</b>	<b>0303</b>	<b><math>PLCP = 2</math></b>	<b>2</b>
3	2	<b>10303</b>	<b>03</b>	<b><math>PLCP = 0</math></b>	<b>0</b>
4	0	<b>1710303</b>	<b>10303</b>	<b><math>PLCP = 1</math></b>	<b>1</b>
5	4	<b>303</b>	<b>1710303</b>	<b><math>PLCP = 0</math></b>	<b>0</b>
6	6	<b>3</b>	<b>303</b>	<b><math>PLCP = 1</math></b>	<b>1</b>
7	1	<b>710303</b>	<b>3</b>	<b><math>PLCP = 0</math></b>	<b>0</b>

**Arreglo LCP** (Indexado por rango  $i$  de 0 a 7):

$$LCP = [0, 0, 2, 0, 1, 0, 1, 0]$$

## Pregunta 2

Este problema es una aplicación directa del algoritmo de pre-procesamiento utilizado en Knuth-Morris-Pratt (KMP), el cual calcula la longitud del Prefijo Propio más Largo (LPS, por sus siglas en inglés) que también es un sufijo para cada prefijo de la cadena.

El algoritmo KMP utiliza una tabla de saltos ( $b$  o  $\pi$ ) que, para una cadena  $x$ , almacena en la posición  $i$  la longitud del prefijo propio más largo de  $x[0..i]$  que también es un sufijo de  $x[0..i]$ .

Si aplicamos este algoritmo de pre-procesamiento a la cadena  $S$ , el último elemento de la tabla LPS calculará precisamente la longitud  $L$  del prefijo propio más largo de  $S$  que es también un sufijo de  $S$ .

La complejidad temporal de este pre-procesamiento es lineal,  $O(N)$ , lo que satisface el requerimiento de eficiencia del problema.

## Implementación en C++

El archivo funcional se encuentra en [lps.cpp](#)

```
1 // Función para calcular la longitud del Longest Proper Prefix (LPS) array para KMP.
2 // La complejidad es O(n).
3 string compute_lps_array(const std::string &S) {
4     int N = S.length();
5     // La subcadena más grande debe ser propia (T != S), así que si N <= 1,
6     // solo podemos retornar la cadena vacía.
7     if (N <= 1)
8         return "";
9
10    // lps[i] almacenará la longitud del prefijo propio más largo de S[0...i]
11    // que también es un sufijo de S[0...i].
12    vector<int> lps(N, 0);
13
14    int len = 0; // Longitud del prefijo más largo anterior que es también sufijo
15    int i = 1;
16
17    // Lazo que calcula lps[i] para i = 1 a N-1
18    while (i < N) {
19        if (S[i] == S[len]) {
20            len++;
21            lps[i] = len;
22            i++;
23        } else {
24            // Desigualdad de caracteres
25            // Retrocedemos a la longitud LPS anterior (len - 1)
26            if (len != 0)
27                len = lps[len - 1];
28            else {
29                // Si len es 0, no hay un prefijo común, lps[i] = 0
30                lps[i] = 0;
31                i++;
32            }
33        }
34    }
35    // La longitud L de la subcadena T deseada es el último elemento del arreglo LPS.
36    // Esto se debe a que lps[N-1] almacena la longitud del prefijo más largo
37    // de S[0...N-1] que también es un sufijo de S[0...N-1].
38    int L = lps[N - 1];
39
40    // Extraemos la subcadena de longitud L.
41    return S.substr(0, L);
42 }
```