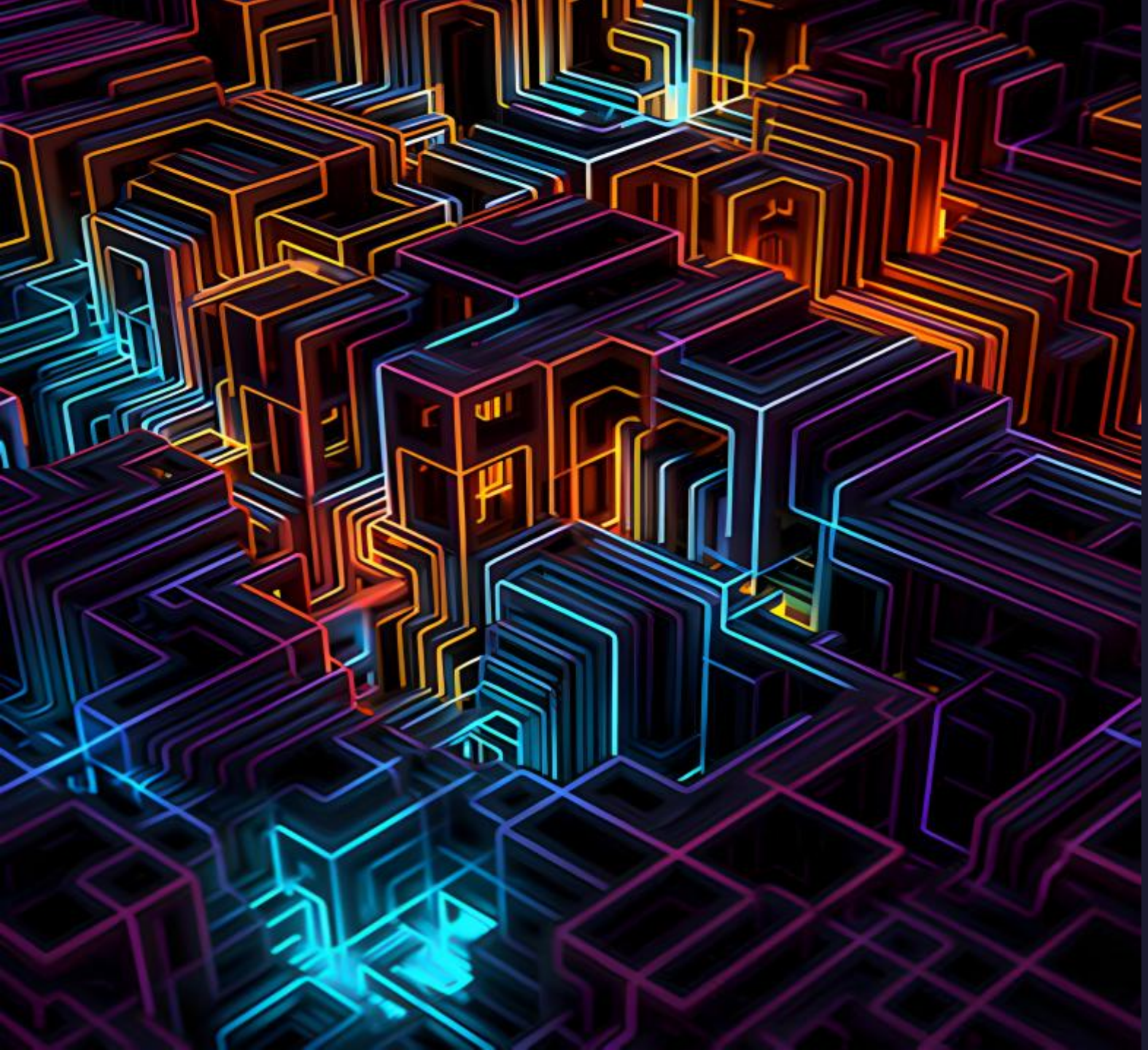


Procesamiento de strings

Análisis y diseño de algoritmos avanzados

Dra. Valentina Narváez Terán

Tecnológico de Monterrey

Problema: string matching

t	h	i	s	_	i	s	_	a	_	s	m	a	l	l	_	e	x	a	m	p	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22

El texto

Dado un texto de tamaño n, y un patrón de tamaño m, encuentra los índices de un substring del texto que contenga el patrón

s	m	a	l	l	_	e	x	a	m	p	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12

El patrón

Para este ejemplo, la salida son los índices 10 y 22

¿Cómo seria el algoritmo de fuerza bruta?
¿Qué complejidad tendría?

Algoritmo naive (ingenuo)

Algoritmo naive para string-matching

$n =$ longitud del texto
$m =$ longitud del patrón

$$
\begin{aligned}
&for\ i = 0\ to\ n - m \\
&\quad matches = 0 \\
&\quad for\ j = 0\ to\ m \\
&\quad\quad if\ patron[i] = texto[i] \\
&\quad\quad\quad matches + + \\
&\quad\quad else \\
&\quad\quad\quad break
\end{aligned}
$$

Si $matches == m$, el patrón esta a partir de i

Es un algoritmo de fuerza bruta (lo implementaste en la actividad 1.1)

Tiene complejidad $f\left(n - m(m)\right) \in O(n^2)$

Es común encontrarlo en la web como $O(n * m)$, por lo que es un ejemplo de complejidades cuadráticas enmascaradas como aparentemente lineales

Algoritmo naive (o ingenuo): un mal caso

$texto$	e	f	f	a	b	l	e	_	e	f	f	a	n	i	n	e	f	f	a	b	l	e
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

$patron$	e	f	f	a	n	i	n	e	f	f	a	b	l	e
	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Consideremos este ejemplo de texto y patrón.

Los caracteres 0 a 3 coindicen. ¿Qué pasa cuando $texto[4]$ y $patron[4]$ no coincidan?

¿Cuáles serán las siguientes posiciones a comparar?

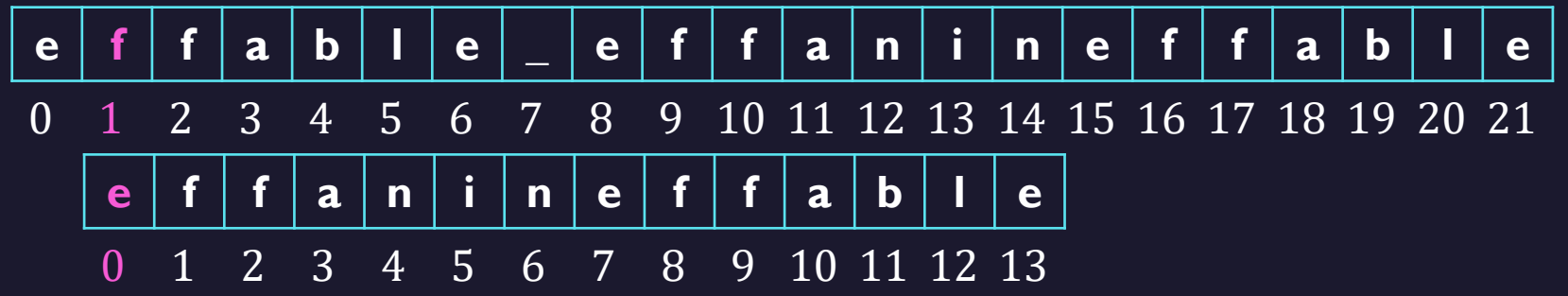
The naming of cats (fragmento)

When you notice a cat in profound meditation,
The reason, I tell you, is always the same:
His mind is engaged in a rapt contemplation
Of the thought, of the thought, of the thought of his name:
His ineffable effable effanineffable
Deep and inscrutable singular name.

T. S. Eliot

Algoritmo naive (o ingenuo): un mal caso

El algoritmo naive reiniciaría la comparación a partir de $texto[1]$ y $patron[0]$



Y cuando no coincidan, reinicia a partir de $texto[2]$ y $patron[0]$

¿Un algoritmo mejor?

Quizá tu primera idea para un algoritmo mas eficiente sea:
si $texto$ y $patron$ coincidieron en los previos 4 caracteres,
seguir comparando solo a partir de $texto[4]$ y $patron[0]$

e	f	f	a	b	l	e	_	e	f	f	a	n	i	n	e	f	f	a	b	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

e	f	f	a	n	i	n	e	f	f	a	b	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Y eventualmente…

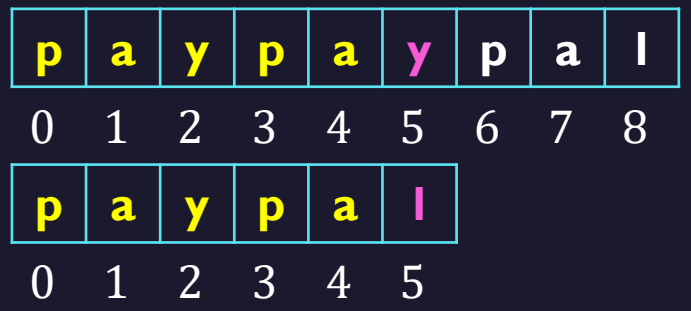
e	f	f	a	b	l	e	_	e	f	f	a	n	i	n	e	f	f	a	b	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

e	f	f	a	n	i	n	e	f	f	a	b	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13

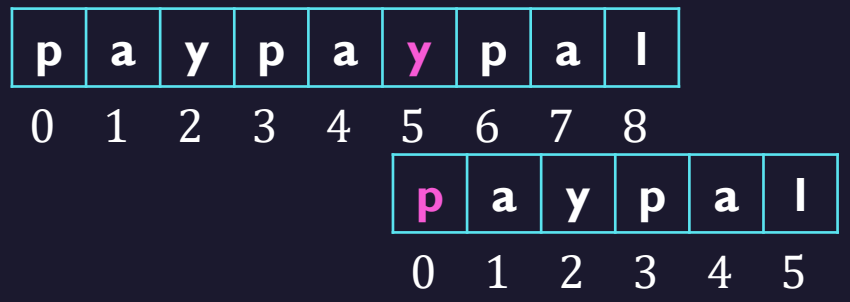
Pero ese enfoque tiene un problema ¿cuál es?

¿Un algoritmo mejor?

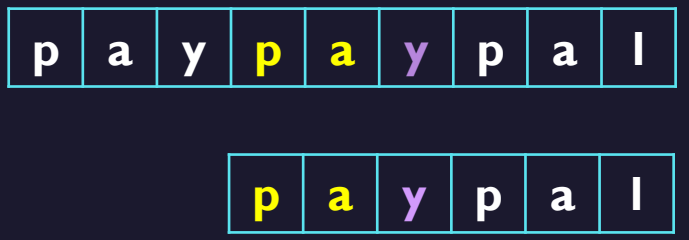
Si ese enfoque se aplica a este ejemplo, nunca se encontrara paypal



La siguiente comparación comenzaría desde $texto[5]$ y $patron[0]$



Esto ocurre porque "paypal" tiene subcadenas que se traslapan con respecto a si misma



Así que convendría conocer sus longitudes para desplazar el patrón solo lo adecuado

Es decir, retomar la comparación desdela y $texto[5]$ y $patron[2]$

Algoritmo Knuth–Morris–Pratt (KMP)

El algoritmo KMP:

- toma ventaja de comparaciones previas y subcadenas traslapadas del patrón
- solo revisar cada posición del texto una única vez.

Desarrollado en los 70's por: James H. Morris, Donald Knuth y Vaughan Pratt

KMP utiliza una tabla LPS (longest prefix sufix), con el precálculo de las longitudes de los prefijos mas largos que también son sufijos a partir de cada índice del patrón

	p	a	y	p	a	l
LPS	0	0	0	1	2	0
index	0	1	2	3	4	5

"pa"es el prefijo mas largo que a la vez es un sufijo de "paypal"

Su longitud es 2. Se guarda a partir de donde se encuentra el sufijo

Algoritmo Knuth–Morris–Pratt (KMP)

¿Cómo seria el LPS de $effaninefffable$?

patron	e	f	f	a	n	i	n	e	f	f	a	b	l	e
LPS														
index	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Algoritmo Knuth–Morris–Pratt (KMP)

¿Cómo seria el LPS de $effaninefffable$?

$patron$	e	f	f	a	n	i	n	e	f	f	a	b	l	e
LPS	0	0	0	0	0	0	0	1	2	3	4	0	0	1
$index$	0	1	2	3	4	5	6	7	8	9	10	11	12	13

Por ahora, obviemos como se computa LPS, y veamos como se usa en KMP

Algoritmo Knuth–Morris–Pratt (KMP)

Volviendo al punto donde $texto[i]$ y $patron[j]$ no coinciden

i	0	1	2	3	4	5	6	7	8
	p	a	y	p	a	y	p	a	l

	p	a	y	p	a	l
LPS	0	0	0	1	2	0
j	0	1	2	3	4	5

Necesitamos desplazar el patrón (cambiando el valor de j)

j = la longitud del prefijo-sufijo mas largo en la sección donde hubo coincidencias

Es decir $j = 2$

i	0	1	2	3	4	5	6	7	8
	p	a	y	p	a	y	p	a	l

	p	a	y	p	a	l
LPS	0	0	0	1	2	0
j	0	1	2	3	4	5

Algoritmo Knuth–Morris–Pratt (KMP)

$n, m =$ longitudes de $texto$ y $patron$
$i, j = 0$ son índices para iterar en $texto$ y $patron$

Mientras $i < n$
    Si $texto[i]$ y $patron[j]$ son iguales
        $i++$
        $j++$
    Si no
        Si $j > 0$
            $j = LPS[j-1]$
        Si no
            $i++$

    Si $j == m$:
        $patron$ existe a partir de $i - m$
        $j = LPS[j-1]$

Cuando son iguales, los índices i y j avanzan

Cuando no, j se reinicia con la longitud del LPS
Excepto si $j = 0$, en ese caso no hay LPS
previo, así que aumenta i

Si j alcanza a m, tenemos una coincidencia completa.
Para obtener todas las ocurrencias, reseteamos j

¿Cómo obtener el arreglo LPS?

Investiga como obtener eficientemente el arreglo LPS

Es un algoritmo muy breve, con complejidad $O(m)$

Algoritmo Z (Z-function / algorithm)

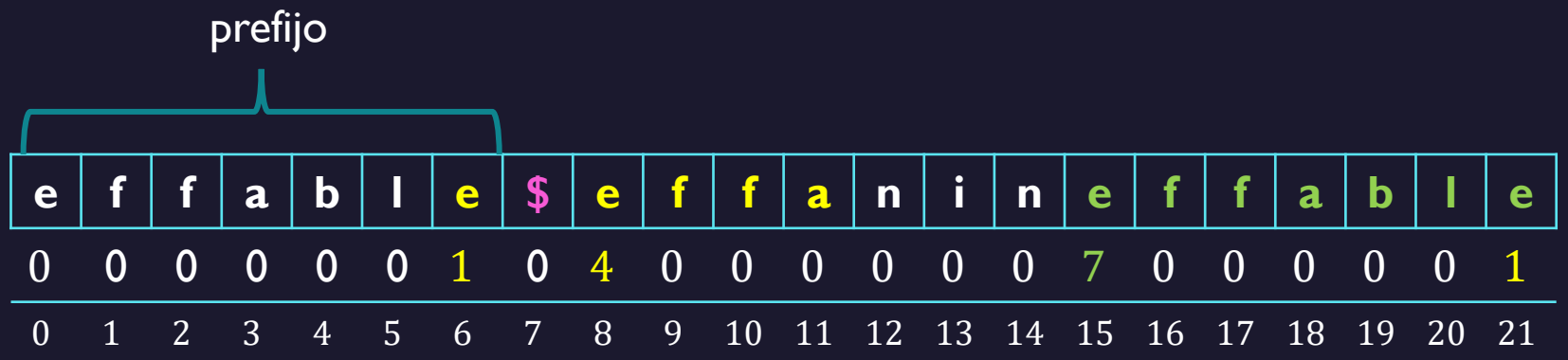
- Es un algoritmo eficiente para string-matching
- Complejidad lineal (si se implementa adecuadamente)

- Usa un arreglo precomputado, llamado Z-array, de tamaño $m + n + 1$

El Z-array se calcula a partir de concatenar patrón y texto, con un separador en medio

e	f	f	a	b	l	e	$	e	f	f	a	n	i	n	e	f	f	a	b	l	e
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21

Algoritmo Z (Z-function / algorithm)



Cada elemento del Z-array indica el numero de coincidencias con el prefijo a partir del índice i

Una vez calculado el Z-array, si $Z[i] == m$, el patrón se ha encontrado en $texto[i - m + 1]$

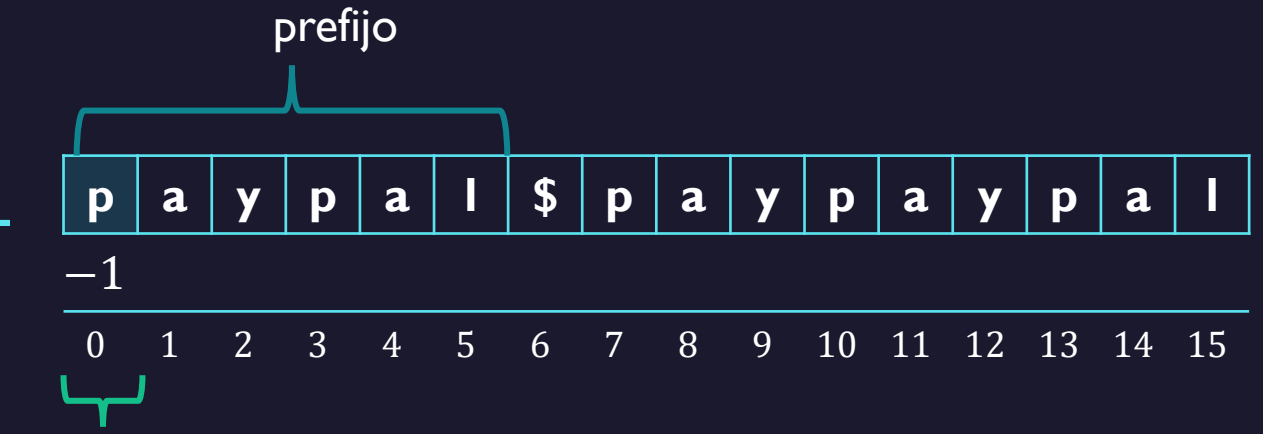
El problema es ¿como calcular Z eficientemente?

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

¿Qué son L y R? Índices inicial y final de una ventana de coincidencias con parte del prefijo. Al inicio, son 0's



Primer iteración: $i = 0$, i esta dentro de la ventana de $L = 0$ a $R = 0$

Copiamos datos de $Z[0]$ a $Z[0]$ (no cambia)

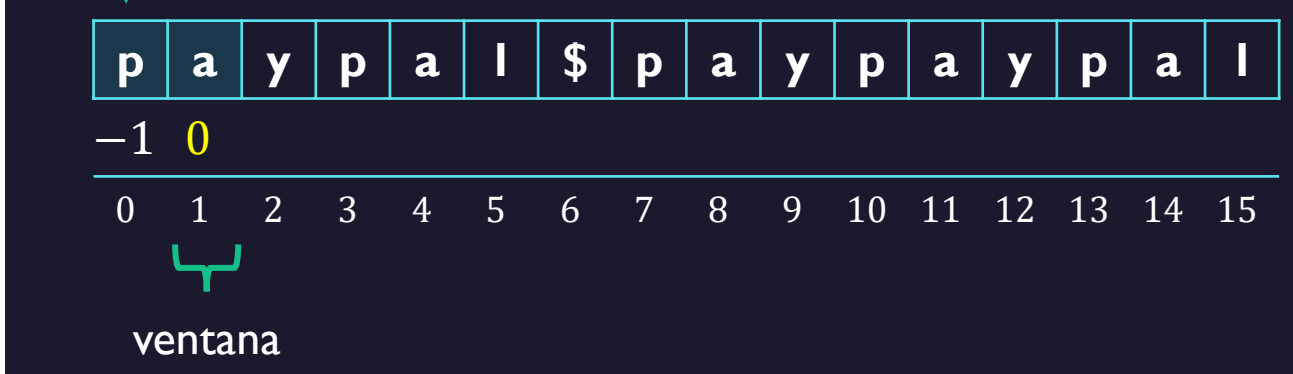
Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

$i = 1$ y $R = 0$

Cuando i esta fuera de los limites de la ultima ventana, R y L se igualan a i



Y se buscan coincidencias nuevas desde $C[R - L]$ y $C[R]$
El valor acumulado de las coincidencias se guarda en $Z[i]$

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

$i = 2$ y $R = 0$

Como i esta fuera de los limites de la ultima ventana, $R = L = i = 2$

Se buscan coincidencias nuevas desde $C[R - L]$ y $C[R]$
El valor acumulado de las coincidencias se guarda en $Z[i]$

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R−−
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R−−
```

$i = 3$ y $R = 3$

Como i esta fuera de los limites de la ultima ventana, $R = L = i = 3$

Se buscan coincidencias nuevas desde $C[R - L]$ y $C[R]$
Ahora $L = 3$ y $R = 5$
Z[i] = 2
R-- es 4

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

Cuando i esta dentro de los limites de la ventana de coincidencias, podemos copiar valores de elementos previos de Z

Pero solo si no estamos dentro de los limites de una ventana anterior

$i = 4$

$k = i - L = 4 - 3 = 1$

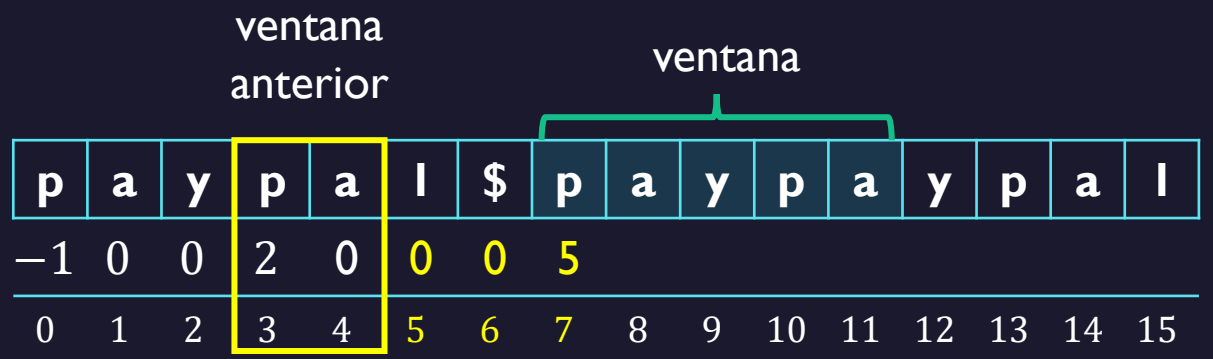
$R - i + 1 = 4 - 4 + 1 = 1$

Las coincidencias a partir de $i = 4$ se pueden copiar de $k = 1$

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R−−
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R−−
```



Cuando i esta fuera de los limites de la ultima ventana (o nunca hemos encontrado una), simplemente se buscan coincidencias nuevas desde $C[R - L]$ y $C[R]$

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

Cuando i esta dentro de los limites de la ventana, o vamos iniciando, podemos copiar valores de elementos previos de Z

Pero solo si no estamos dentro de los limites de una ventana anterior

Si hay 5 coincidencias desde $C[7]$, $Z[8]$ y $Z[9]$ tendrán que coincidir con $Z[1]$ y $Z[3]$

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

Si no, ¿Cómo sabemos que hay traslape con una ventana anterior?

$Z[k] = 2$
$R - i + 1 = 11 - 10 + 1 = 0$

p	a	y	p	a	l	$	p	a	y	p	a	y	p	a	l
−1	0	0	2	0	0	0	5	0	0	?	?				
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

La ventana actual no tiene suficiente información sobre la ventana previa. Visualmente, la ventana actual (rosa) no llega mas allá de la anterior (amarilla).

Entonces hay que buscar coincidencias nuevas desde $C[10]$

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

Buscando coincidencias nuevas desde $C[10]$ se obtiene:

¿Y el resto?

Cálculo del Z-array

L y R: inicio y fin de la ventana de coincidencias
x = $n + m + 1$
Z = arreglo de tamaño x, inicia en $-1's$
C = patrón + "$" + texto

```
for i = 0 to x
    if i ≤ R
        k = i − L
        if Z[k] < R − i + 1
            Z[i] = Z[k]
        else
            L = i
            while R < x and C[R − L] == C[R]
                R++
            Z[i] = R − L
            R − −
    else
        L = R = i
        while R < x and C[R − L] == C[R]
            R++
        Z[i] = R − L
        R − −
```

Para el resto, se puede copiar de los resultados anteriores.

La ventana traslapada anterior no es problema, porque la nueva se extiende mas allá de ella

Uso del Z-array

p	a	y	p	a	l	$	p	a	y	p	a	y	p	a	l
−1	0	0	2	0	0	0	5	0	0	6	0	0	2	0	0
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

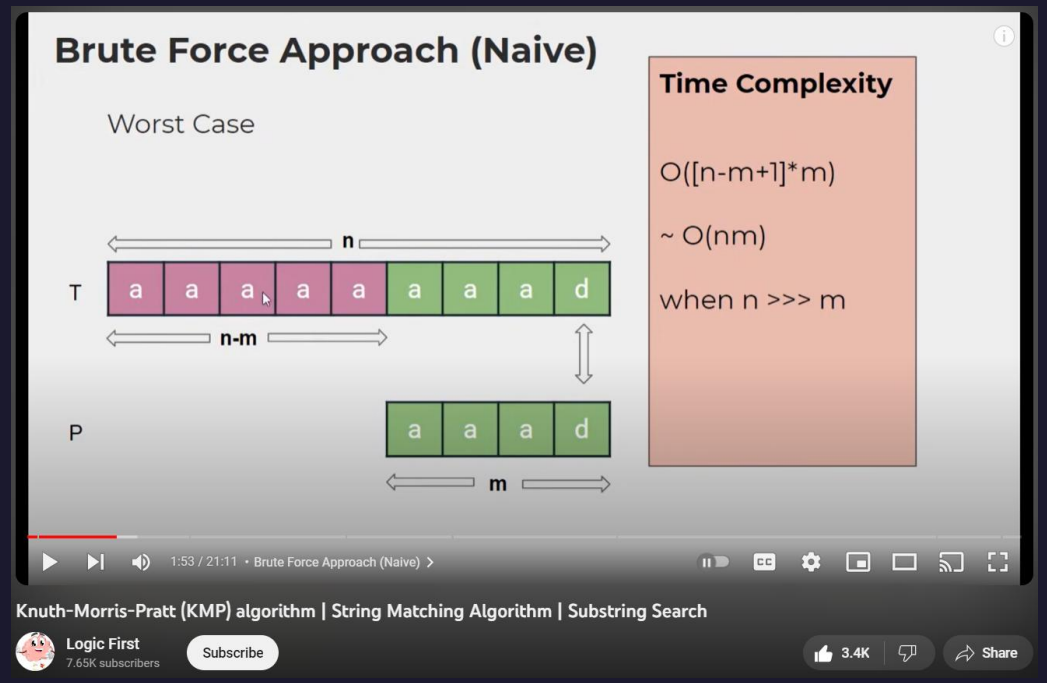
Al final, en donde encontremos el registro de una ventana de tamaño m en el $Z - array$, se encuentra el patrón

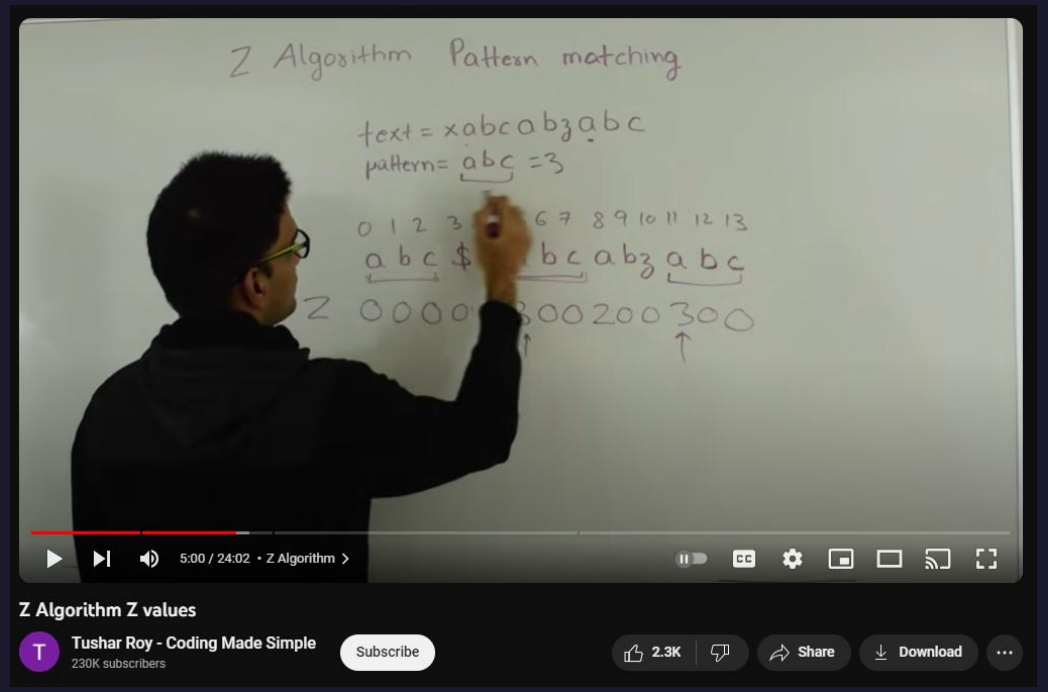
El índice será $i - (m + 1)$ en el texto original (sin concatenar)

p	a	y	p	a	y	p	a	l
0	1	2	3	4	5	6	7	8

$10 - (6 + 1) = 3$

Extras: demos de LPS y Z-array





https://youtu.be/4jY57Ehc14Y?si=sY2weauoI_b2qyDp

https://youtu.be/CpZh4eF8QBw?si=dItSF_JMvAxfXHpP