



Textos y Cadenas

Búsqueda de patrones

búsqueda de cadenas

La búsqueda de cadenas (string matching) o búsqueda de patrones (pattern matching) es un tipo de algoritmos destinados a buscar una subcadena o patrón dentro de una cadena

búsqueda de cadenas: aplicaciones

- Procesadores de texto
- Utilidades de OS: grep, ack, ag
- Bases de datos
- Búsqueda en la Web
- Biología: secuencias de ADN

grep usa el algoritmo de boyer-moore.

Ver: <https://lists.freebsd.org/pipermail/freebsd-current/2010-August/019310.html>

búsqueda de cadenas: nomenclatura

Llamaremos

n a la longitud del texto o cadena en que se buscará la subcadena o patrón

m a la longitud de la subcadena o patrón

algoritmo de fuerza bruta

El algoritmo naive o de fuerza bruta desplaza la subcadena un carácter por cada intento y compara el patrón y la porción de texto correspondiente carácter a carácter hasta que encuentra una no coincidencia o llega al final del patrón.

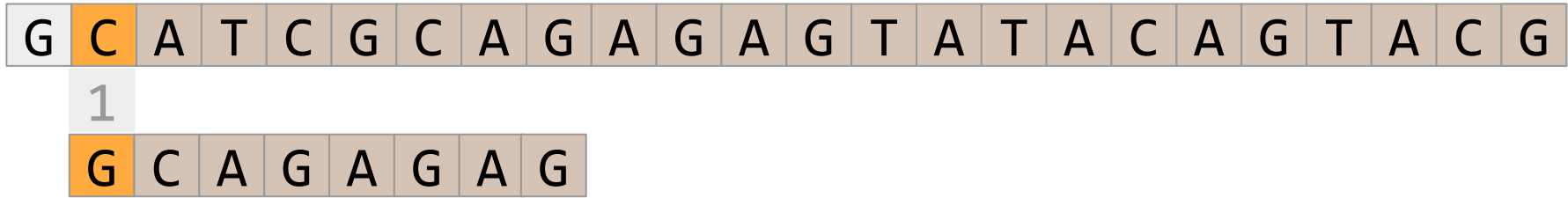
algoritmo de fuerza bruta

Primer intento

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
1	2	3	4																				
G	C	A	G	A	G	A	G																

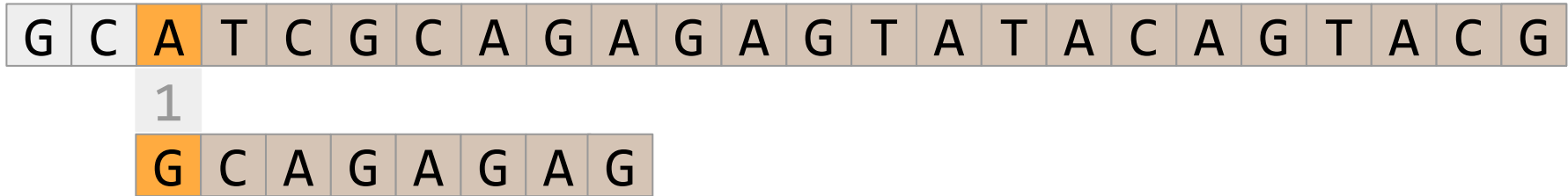
algoritmo de fuerza bruta

Segundo intento



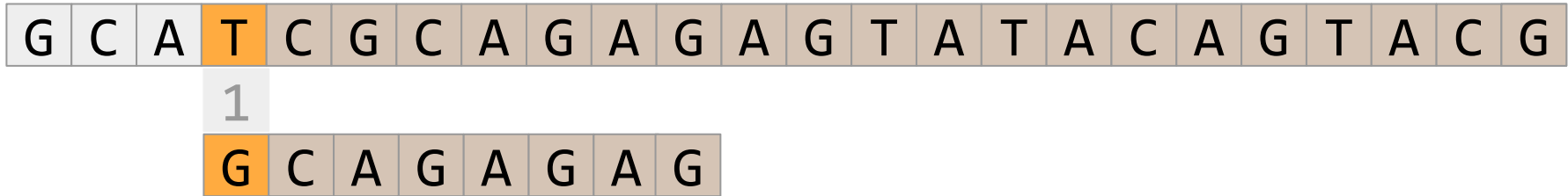
algoritmo de fuerza bruta

Tercer intento



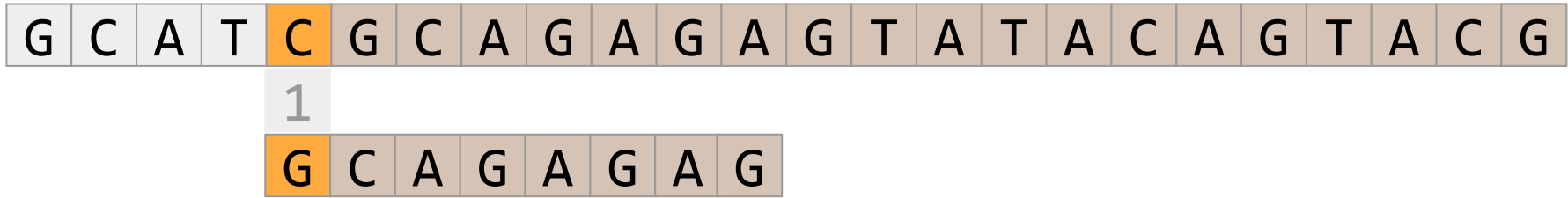
algoritmo de fuerza bruta

Cuarto intento



algoritmo de fuerza bruta

Quinto intento



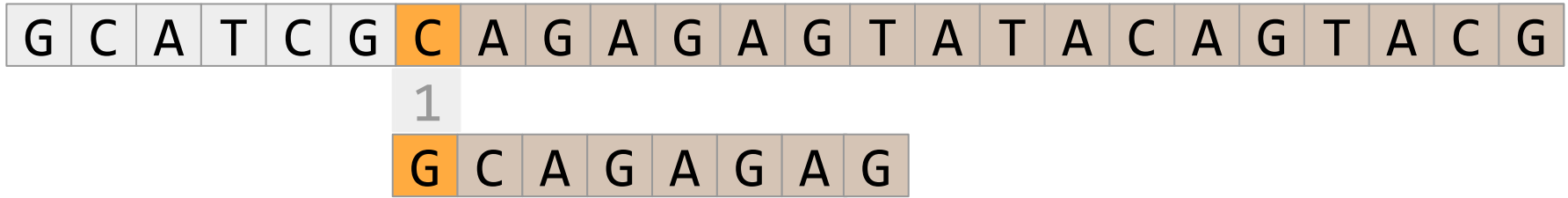
algoritmo de fuerza bruta

Sexto intento

G	C	A	T	C	G	C	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C	G
					1	2	3	4	5	6	7	8											
					G	C	A	G	A	G	A	G											

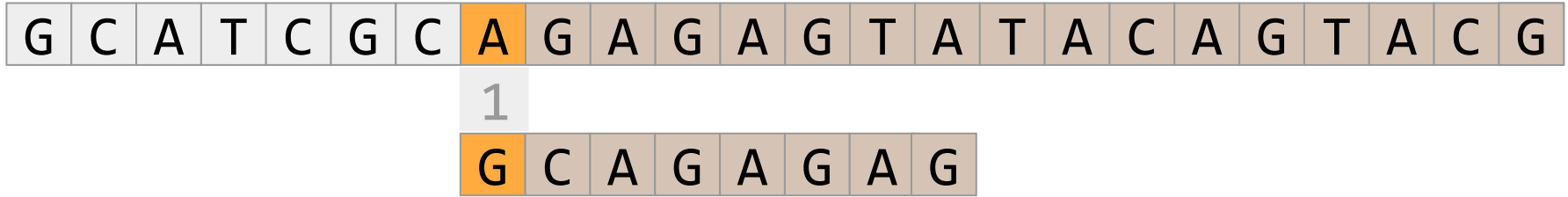
algoritmo de fuerza bruta

Séptimo intento



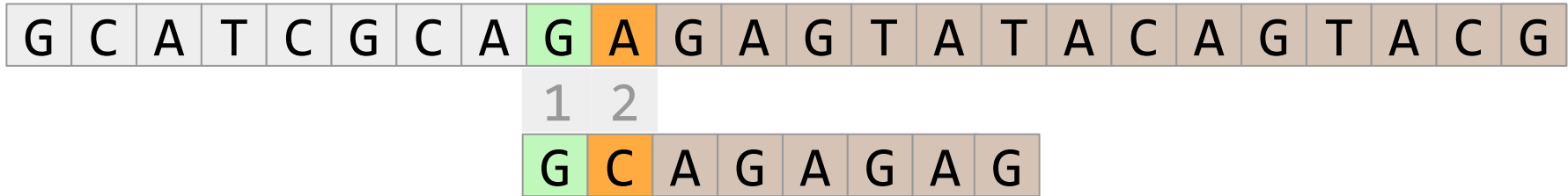
algoritmo de fuerza bruta

Octavo intento



algoritmo de fuerza bruta

Noveno intento



algoritmo de fuerza bruta

Para este caso realiza 30 comparaciones en 17 intentos

eficiencia

En el peor de los escenarios el algoritmo tiene complejidad **$O(m(n-m+1))$**

La eficiencia es **$O(nm)$**



Ejercicio 5.01.

Implementá el algoritmo por fuerza bruta para buscar un patrón en un texto

algoritmo Knuth - Morris - Pratt (KMP)

Este algoritmo trata de mejorar el algoritmo de fuerza bruta evitando algunas comparaciones innecesarias.

Para eso busca dentro del patrón sufijos que también son prefijos y cuando se detecta una no coincidencia después de un prefijo, en vez de desplazar un carácter se desplaza hasta la posición del prefijo

algoritmo Knuth - Morris - Pratt (KMP)

Para esto se realiza un pre procesamiento, antes de comenzar la búsqueda propiamente dicha, con el cual obtenemos la tabla de fallos o tabla de sufijos propios.

Para este pre procesamiento se utiliza solamente el patrón.

algoritmo Knuth - Morris - Pratt (KMP)

Tabla de prefijos: ejemplos

p[]	A	B	C	E	A	B	D	A	B	F
	0	1	2	3	4	5	6	7	8	9
f[]	0	0	0	0	1	2	0	1	2	0

p[]	A	A	B	C	A	D	A	A	B	E
	0	1	2	3	4	5	6	7	8	9
f[]	0	1	0	0	1	0	1	2	3	0

p[]	A	B	C	D	E	A	B	F	A	B	C
	0	1	2	3	4	5	6	7	8	9	10
f[]	0	0	0	0	0	1	2	0	1	2	3

p[]	A	A	A	A	B	A	A	C	D	A
	0	1	2	3	4	5	6	7	8	9
f[]	0	1	2	3	0	1	2	0	0	1

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]								

El array p es el patrón. El array f es la tabla de fallos o prefijos.

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j	i						
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0							

Partimos de esta situación inicial. Como $p[j] \neq p[i] \Rightarrow f[i] = 0$
Incrementamos i

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j		i					
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0						

Nuevamente $p[j] \neq p[i] \Rightarrow f[i] = 0$
Incrementamos i

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j		i					
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0	0					

$p[j] \neq p[i] \Rightarrow f[i] = 0$
Incrementamos i

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j				i			
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0	0	0				

$p[j] == p[i] \Rightarrow f[i] = j + 1$
Incrementamos i y j

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j				i			
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0	0	0	1			

$p[j] == p[i] \Rightarrow f[i] = j + 1$
Incrementamos i y j

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

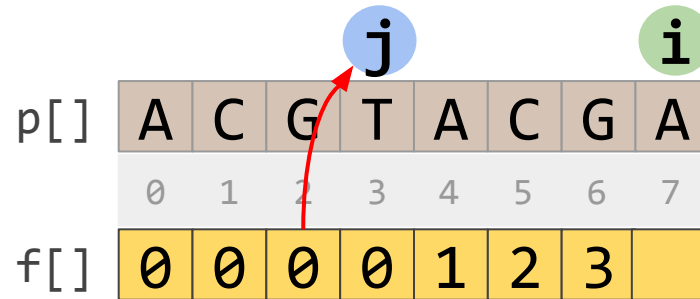
			j				i	
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0	0	0	1	2		

$p[j] == p[i] \Rightarrow f[i] = j + 1$
Incrementamos i y j

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j				i			
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0	0	0	1	2	3	



$$p[j] \neq p[i] \Rightarrow j = f[j-1]$$

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j							i
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0	0	0	1	2	3	

$$p[j] == p[i] \Rightarrow f[i] = j + 1$$

algoritmo Knuth - Morris - Pratt (KMP)

Obtener la tabla de prefijos

	j							i
p[]	A	C	G	T	A	C	G	A
	0	1	2	3	4	5	6	7
f[]	0	0	0	0	1	2	3	1

fin



Ejercicio 5.02.

Encontrá la tabla de fallos para el algoritmo de Knuth - Morris - Pratt de los siguientes patrones

A	A	B	A	A	B	A	A	A
0	1	2	3	4	5	6	7	8

A	B	A	B	C	A	B	A	B
0	1	2	3	4	5	6	7	8



Ejercicio 5.02.

Encontrá la tabla de fallos para el algoritmo de Knuth - Morris - Pratt de los siguientes patrones

A	A	B	A	A	B	A	A	A
0	1	2	3	4	5	6	7	8
0	1	0	1	2	3	4	5	2

A	B	A	B	C	A	B	A	B
0	1	2	3	4	5	6	7	8
0	0	1	2	0	1	2	3	4

algoritmo Knuth - Morris - Pratt (KMP)

Algoritmo de búsqueda

A diferencia del algoritmo de fuerza bruta en KMP la idea es saltar aquellas comparaciones que ya sabemos de antemano que no arrojarán resultado positivo.

Esto lo hacemos utilizando la información que tenemos en la tabla de fallos (o sufijos propios)

algoritmo Knuth - Morris - Pratt (KMP)

Algoritmo de búsqueda

Comenzamos comparando `patron[j]` para $j = 0$ con los caracteres correspondientes del texto

Mientras los caracteres `texto[i]` coinciden con `patron[j]` incrementamos i y j

Cuando no coinciden:

Sabemos que `patron[0..j-1]` coincide con `texto [i-j+1...i-1]`

Tambien que `f[j-1]` tiene la cantidad de caracteres de `patron[0..j-1]` que son prefijos y sufijos propios

Entonces no necesitamos comparar los `f[j-1]` caracteres con `texto[i-j.., i-1]` porque ya sabemos que coinciden

Knuth - Morris - Pratt

Primer intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
0	1	2	3	4																			
G	A	G	A	G																			
0	0	1	2	3																			

Knuth - Morris - Pratt

Segundo intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
		0	1	2																			
		G	A	G	A	G																	
		0	0	1	2	3																	

Knuth - Morris - Pratt

Tercer intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
				∅																			
				G	A	G	A	G															
				∅	∅	1	2	3															

Knuth - Morris - Pratt

Cuarto intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
					∅																		
					G	A	G	A	G														
					∅	∅	1	2	3														

Knuth - Morris - Pratt

Quinto intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
						0	1	2															
						G	A	G	A	G													
						0	0	1	2	3													

Knuth - Morris - Pratt

Sexto intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
								0															
								G	A	G	A	G											
								0	0	1	2	3											

Knuth - Morris - Pratt

Septimo intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
									0	1	2	3	4										
									G	A	G	A	G										
									0	0	1	2	3										

eficiencia

Procesar la tabla de fallos tiene eficiencia **$O(m)$**

La búsqueda se realiza en **$O(n)$**

La eficiencia del algoritmo es **$O(n+m)$**



Ejercicio 5.03.

Implementá el algoritmo para buscar un patrón en un texto usando el algoritmo de Knuth - Morris - Pratt

algoritmo de Boyer - Moore - Horspool

Este algoritmo implica una mejora de KMP para alfabetos largos

Es una simplificación del algoritmo de Boyer - Moore

Compara de derecha a izquierda

Hay un preprocesamiento del patrón para obtener la tabla de mal caracter

Boyer - Moore - Horspool

Pre procesamiento

Si c está en el patrón:

$l[c] = \text{longitud} - \text{indice} - 1$

Si no:

$l[c] = \text{longitud}$

La última letra del patrón se ignora

G	A	G	A	G
0	1	2	3	4

l[]	G	A	*
	2	1	5

Boyer - Moore - Horspool

Primer intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
0	1	2	3	4																			
G	A	G	A	G																			

1[]	G	A	*
	2	1	5

No hay coincidencia $1[T] = 5 \Rightarrow$ desplazamos 5

Boyer - Moore - Horspool

Segundo intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
						0	1	2	3	4													
						G	A	G	A	G													

1[]	G	A	*
	2	1	5

No hay coincidencia $1[A] = 1 \Rightarrow$ desplazamos 1

Boyer - Moore - Horspool

Tercer intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
						0	1	2	3	4													
						G	A	G	A	G													

1[]	G	A	*
	2	1	5

No hay coincidencia $1[A] = 1 \Rightarrow$ desplazamos 1

Boyer - Moore - Horspool

Cuarto intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
							0	1	2	3	4												
							G	A	G	A	G												

1[]	G	A	*
	2	1	5

No hay coincidencia $1[A] = 1 \Rightarrow$ desplazamos 1

Boyer - Moore - Horspool

Quinto intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
								0	1	2	3	4											
								G	A	G	A	G											

1[]	G	A	*
	2	1	5

No hay coincidencia $1[A] = 1 \Rightarrow$ desplazamos 1

Boyer - Moore - Horspool

Sexto intento

G	A	G	A	T	C	G	A	A	G	A	G	A	G	T	A	T	A	C	A	G	T	A	C
									0	1	2	3	4										
									G	A	G	A	G										

1[]	G	A	*
	2	1	5

Encontrada

eficiencia

La eficiencia del algoritmo es **$O(nm)$ en el peor caso**. La misma que el algoritmo naive

En el mejor caso **$O(m/n)$**



Ejercicio 5.04.

Implementá el algoritmo para buscar un patrón en un texto usando el algoritmo de Boyer Moore Hoorspool

comparación de eficiencia

