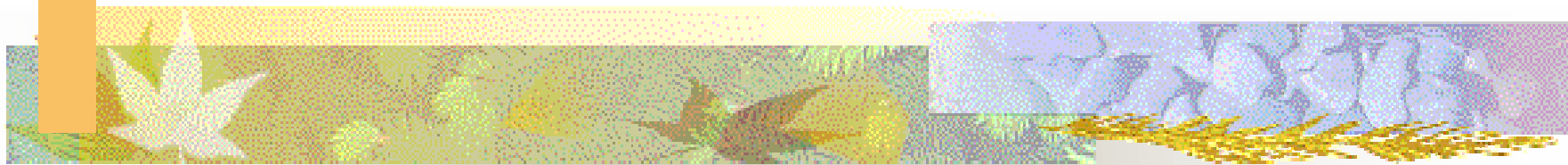


Algoritmos de Búsqueda Secuencial de Texto



Universidad de Costa Rica

Maestría en Computación e Informática

Kryscia Daviana Ramírez Benavides

993237



Agenda

- Introducción.
- Algoritmo Fuerza Bruta (Naive).
- Algoritmo No tan Fuerza Bruta (Not So Naive).
- Algoritmo Karp-Rabin.
- Algoritmo Shift-Or.
- Algoritmo Knut-Morris-Pratt.
- Algoritmo Boyer-Moore y Boyer-Moore-Horspool.
- Conclusiones.
- Referencias Bibliográficas.



Introducción

- Aunque los datos se pueden presentar de varias maneras, el texto sigue siendo la forma principal para intercambiar la información:
 - En la literatura o la lingüística los datos se componen de la recopilación y de diccionarios enormes.
 - En la informática existe una gran cantidad de datos que se almacenan en archivos.
 - En la biología molecular, las moléculas biológicas se pueden aproximar a menudo como secuencias de nucleótidos o de aminoácidos.



Introducción (cont.)

- Además, la cantidad de datos disponibles en estos campos tiende a duplicarse cada 18 meses.
- Ésta es la razón por la que los algoritmos deben ser eficientes, incluso la velocidad y la capacidad del almacenaje de computadoras aumentan regularmente.
- El **búsqueda secuencial de texto** consiste en encontrar una o, más generalmente, ocurrencias de una secuencia (generalmente llamada **patrón**) en un **texto**.



Introducción (cont.)

■ Algoritmos más comunes:

- Fuerza bruta.
- Familia de algoritmos de corrimiento.
 - Permite análisis (o búsqueda) paralelo.
 - Permite encontrar de una sola pasada dos ocurrencias de la subhilera ABRACADABRA en la hilera **ABRACADABRACADABRA**
 - Funciona como una máquina de estados.
- Familia de algoritmos Boyer–Moore.
 - No analiza todos los caracteres del texto.
 - Preprocesa el patrón a buscar para calcular saltos en el análisis (o búsqueda).



Introducción (cont.)

- Todos los algoritmos que se exponen, encuentran todas las ocurrencias del **patrón** en el **texto**.
- El **patrón** se denota por $x = x[0, \dots, m-1]$; su longitud es igual a m .
- El **texto** se denota por $y = y[0, \dots, n-1]$; su longitud es igual a n .
- Ambas secuencias son estructuras sobre un sistema finito de caracteres llamado **alfabeto** denotado por Σ , con tamaño igual a σ .



Introducción (cont.)

- En cada algoritmo se presenta:
 - Las características principales.
 - La descripción.
 - El código C.
 - El comportamiento con un ejemplo típico, donde:
 - $x = \text{GCAGAGAG}$.
 - $y = \text{GCATCGCAGAGAGTATACAGTACG}$.

Algoritmo Fuerza Bruta (Ingenuo – Naive)





Principales Características

- No existe un preprocesamiento del patrón.
- Requiere espacio constante.
- Realiza siempre saltos de un carácter.
- Compara de izquierda a derecha.
- Realiza la búsqueda del patrón en un tiempo $O(mn)$.
- Realiza $2n$ comparaciones previstas de los caracteres del texto.



Descripción

- No requiere ninguna fase de preproceso previo, ni un espacio extra constante además del espacio asignado al patrón y al texto.
- Para la búsqueda:
 - Consiste en la comparación de todas las posiciones del texto entre 0 y el $n-m$, si una ocurrencia del patrón corresponde o no.
 - Si encuentra una no ocurrencia, o una ocurrencia total del patrón, salta un carácter hacia la derecha.



Código C

Ver [Algoritmo-FB.txt](#)



Ejemplo

Ver [Algoritmo-FB.pps](#)

Algoritmo No Tan Fuerza Bruta **(No Tan Ingenuo – Not So Naive)**





Principales Características

- Existe un preprocesamiento del patrón.
- Necesita espacio extra constante y tiempo $O(m)$ (por el preprocesamiento).
- Realiza saltos de uno o dos caracteres (depende del preprocesamiento).
- Compara de izquierda a derecha.
- Realiza la búsqueda del patrón en un tiempo $O(mn)$.
- Realiza $2n$ comparaciones previstas de los caracteres del texto.



Descripción

- Se calcula el preprocesamiento del patrón de la siguiente forma:
 - Si $x[0] = x[1]$ entonces $K = 2$ y $L = 1$.
 - En caso contrario $K = 1$ y $L = 2$.
- Para la búsqueda:
 - Consiste en la comparación de cada carácter del texto con las posiciones del patrón en el orden $1, 2, \dots, m-2, m-1$ y 0 , si se da una ocurrencia del patrón o no.
 - Si encuentra una no ocurrencia, o una ocurrencia total del patrón, se cuenta cuantas comparaciones ha realizado:
 - Si las comparaciones son mayor que 1 entonces salta L caracteres hacia la derecha.
 - En caso contrario salta K caracteres hacia la derecha.



Código C

Ver [Algoritmo-NTFB.txt](#)

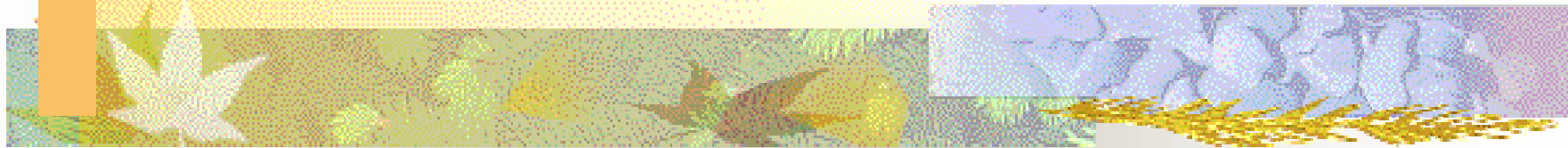


Ejemplo

- Preprocesamiento:
 - Patrón = GCAGAGAG.
 - $x[0] = G$ y $x[1] = C$.
 - $x[0] \neq x[1] \Rightarrow K = 1$ y $L = 2$.

Ver [Algoritmo-NTFB.pps](#)

Algoritmo Karp-Rabin





Principales Características

- Usa una función de **hash**.
- Existe un preprocesamiento del patrón.
- Necesita espacio y tiempo extra constante (por el preprocesamiento).
- Se corre un carácter hacia la derecha.
- Realiza la búsqueda del patrón en un tiempo $O(mn)$.
- Se espera un tiempo de ejecución de $O(n+m)$.



Descripción

- El *hashing* proporciona un método simple para evitar un número cuadrático de comparaciones de caracteres en la mayoría de las situaciones prácticas.
- En vez de comprobar en cada posición del texto si ocurre el patrón, es más eficiente comprobar solamente si el *substring* del texto “parece” el patrón.
- Para comprobar la semejanza entre esos dos *strings* se utiliza una función **hash**.



Descripción (cont.)

- Se calcula el preprocesamiento del patrón de la siguiente forma:
 - Se calcula el $hash(x)$.
- Para la búsqueda:
 - Consiste en la comparación del $hash(x)$ con el $hash(y[j, ..., j+m-1])$ para $0 \leq j < n-m$:
 - Si los **hash** son iguales entonces es necesario comparar carácter por carácter para ver que haya ocurrencia.
 - Se corre un carácter hacia la derecha si no hay ocurrencia o se da una ocurrencia total.



Código C

Ver [Algoritmo-KR.txt](#)

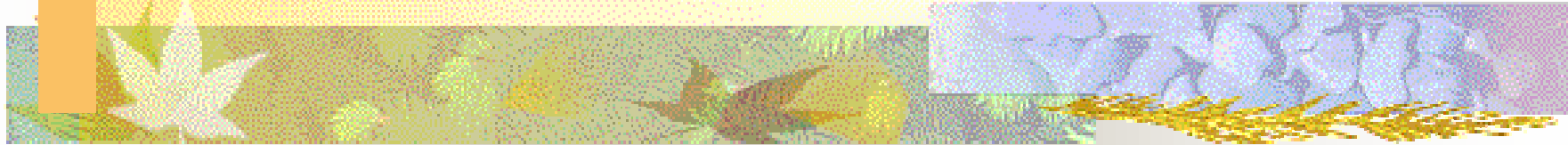


Ejemplo

- Preprocesamiento:
 - Patrón = GCAGAGAG.
 - $hash(x) = 17597$.

Ver [Algoritmo-KR.pps](#)

Algoritmo Shift-Or





Principales Características

- Utiliza una máquina de estados.
- Existe un preprocesamiento del patrón.
- Necesita $O(m+\sigma)$ en espacio y tiempo (por el preprocesamiento).
- Realiza corrimientos de un carácter hacia la izquierda.
- Se corre un carácter hacia la derecha.
- Realiza la búsqueda del patrón en un tiempo $O(n)$ (independiente de la longitud del patrón y el texto).



Descripción

- Este algoritmo trabaja con un conjunto de caracteres, un **alfabeto** y utiliza una **máquina de estados**.
- El **alfabeto** lo forma todos los caracteres diferentes del texto y del patrón.
- Los estados son la comparación de las diferentes partes del patrón con diferentes partes del texto, en donde se utilizan dos operaciones:
 - Corrimientos (*SHIFT*), paso de un estado a otro.
 - Operación lógica *OR*, para calcular el siguiente estado.
- El estado inicial siempre es de sólo 1's (la cantidad depende de la longitud del patrón).



Descripción (cont.)

- Se calcula el preprocesamiento del patrón de la siguiente forma:
 - El patrón se invierte y se indica las apariciones de cada carácter.
 - Para cada carácter del patrón se saca su estado de aparición en la misma.
 - Caracteres iguales se denotan con 0.
 - Caracteres diferentes se denotan con 1.
 - Caracteres que no se encuentren en el alfabeto del patrón tendrán asignados un estado de sólo 1's.



Descripción (cont.)

- Para la búsqueda:
 - Se toma el estado actual, se le hace corrimiento a la izquierda de 1 bit y se le aplica la operación OR con el patrón correspondiente al carácter actual.
 - Se corre un carácter hacia la derecha si no hay ocurrencia o se da una ocurrencia total.



Código C

Ver [Algoritmo-SO.txt](#)

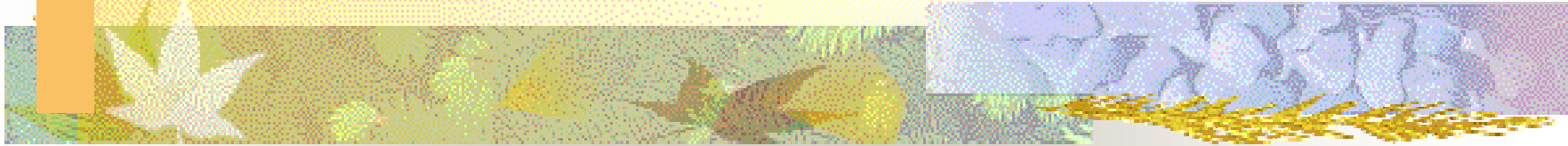
Ejemplo

- Alfabeto:
 - A – C – G – T.
- Preprocesamiento:
 - Patrón = GCAGAGAG.

	G	A	G	A	G	A	C	G
S _A	1	0	1	0	1	0	1	1
S _C	1	1	1	1	1	1	0	1
S _G	0	1	0	1	0	1	1	0
S _T	1	1	1	1	1	1	1	1

Ver [Algoritmo-SO.pps](#)

Algoritmo Knuth-Morris-Pratt





Principales Características

- Existe un preprocesamiento del patrón.
- Necesita $O(m)$ en espacio y tiempo (por el preprocesamiento).
- Realiza saltos determinados en el preprocesamiento.
- Compara de izquierda a derecha.
- Realiza la búsqueda del patrón en un tiempo $O(n+m)$ (independiente de la longitud del patrón y el texto).



Descripción

- Es razonable encontrar que un prefijo v del patrón empareje en algún sufijo de la porción u del texto.
- El más largo prefijo v se llama la **frontera marcada** de u (ocurre en ambos finales de u seguidos por diferentes caracteres en x).
- Esto introduce la notación de $kmpNext[i]$, es la longitud de la frontera más larga de $x[0, \dots, i-1]$ seguida por un carácter c diferente de $x[i]$ y -1 si ninguna etiqueta de la frontera de salida, para $0 < i \leq m$.
- El valor de $kmpNext[0]$ es -1.



Descripción (cont.)

- Se calcula el preprocesamiento del patrón de la siguiente forma:
 - Alinear el segmento, del prefijo, con la ocurrencia más a la izquierda en x que es seguida por un carácter diferente de $x[i]$.
 - Se compara el patrón con él mismo, haciéndolo que empareje el prefijo en alguna parte de la porción del patrón.
 - Al primer carácter del patrón se le asigna el valor de -1, y a los siguientes caracteres si son diferentes a los caracteres del prefijo se calcula a cuantas letras está de distancia del primer carácter del prefijo.



Descripción (cont.)

- Para la búsqueda:
 - Se compara cada carácter del patrón y del texto, de izquierda a derecha.
 - Cuando se encuentra una no ocurrencia o una ocurrencia total, el salto se calcula $i-kmpNext[i]$.
 - En el fondo realiza el mismo proceso que se realizó para realizar el preprocesamiento.



Código C

Ver [Algoritmo-KMP.txt](#)

Ejemplo

■ Preprocesamiento:

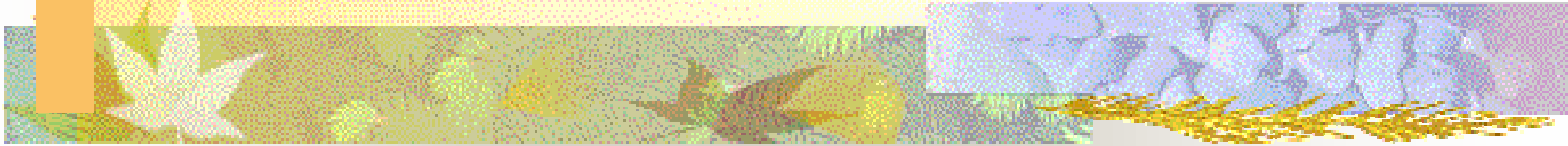
■ Patrón = GCAGAGAG.

<i>i</i>	0	1	2	3	4	5	6	7	8
<i>x</i> [<i>i</i>]	G	C	A	G	A	G	A	G	
<i>kmpNext</i> [<i>i</i>]	-1	0	0	-1	1	-1	1	-1	1

G	C	A	G	A	G	A	G												
G	C	A	G	A	G	A	G												
-1	0	0																	
G	C	A	G	A	G	A	G												
			G	C	A	G	A	G	A	G									
-1	0	0	-1	1															
G	C	A	G	A	G	A	G												
				G	C	A	G	A	G	A	G								
-1	0	0	-1	1	-1	1													
G	C	A	G	A	G	A	G												
					G	C	A	G	A	G	A	G							
-1	0	0	-1	1	-1	1	-1	1											

Ver [Algoritmo-KMP.pps](#)

Algoritmo Boyer-Moore





Principales Características

- Existe un preprocesamiento del patrón.
- Necesita $O(m+\sigma)$ en espacio y tiempo (por el preprocesamiento).
- Realiza saltos determinados en el preprocesamiento.
- Compara de derecha a izquierda.
- Realiza la búsqueda del patrón en un tiempo $O(mn)$.
- Realiza $3n$ comparaciones previstas de los caracteres del texto.
- Mayor desempeño $O(n/m)$.



Descripción

- Es considerado el mejor algoritmo de búsqueda de un patrón en un texto en aplicaciones usuales.
- Los editores de texto utilizan una versión simplificada de este algoritmo para los comandos de búsqueda y reemplazo.
- El algoritmo escanea los caracteres del patrón de derecha a izquierda iniciando con el carácter más a la derecha.
- En caso de una no ocurrencia o una ocurrencia total del patrón se usa dos funciones preprocesadas para saltar hacia la derecha:
 - Salto del **buen sufijo** (*good-suffix shift*) (o salto de *matching*).
 - Salto del **mal carácter** (*bad-character shift*) (o salto de ocurrencia).



Descripción (cont.)

- El salto del buen sufijo consiste en:
 - Alinear el segmento, del sufijo, con la ocurrencia más a la derecha en x que es precedida por un carácter diferente de $x[i]$.
 - Se compara el patrón con él mismo, haciéndolo que empareje el sufijo en alguna parte de la porción del patrón.
 - Si no existe un segmento, el salto consiste en alinear el sufijo largo v con un prefijo de x que haga *matching*.
- El salto del mal carácter consiste en:
 - Alinear cada carácter del **alfabeto** Σ con la ocurrencia más a la derecha en $x[0, \dots, m-2]$.
 - Si el carácter no ocurre en el patrón x , la no ocurrencia de x puede incluir el carácter y alinearlo en el lado más izquierdo del patrón.



Descripción (cont.)

- Se calcula el preprocesamiento del patrón de la siguiente forma:
 - Se calcula las tablas $bmGs$ y $bmBc$.
- Para la búsqueda:
 - Compara los caracteres del patrón y con los caracteres del texto de derecha a izquierda.
 - Cuando se encuentra una no ocurrencia o una ocurrencia total, el salto se calcula $MAX(bmGs[i], bmBc[c]-m+i+1)$.



Código C

Ver [Algoritmo-BM.txt](#)

Ejemplo

- Preprocesamiento:

- Patrón = GCAGAGAG.

i	0	1	2	3	4	5	6	7
$x[i]$	G	C	A	G	A	G	A	G
$suff[i]$	1	0	0	2	0	4	0	8
$bmGs[i]$	7	7	7	2	7	4	7	1

Otro Carácter	G	C	A	G	A	G	A	G
8	7	6	5	4	3	2	1	0

c	A	C	G	T
$bmBc[c]$	1	6	2	8

Ver Algoritmo-BM.pps

1era FASE

$m = 8 \leftarrow$ Tamaño del patrón

G	C	A	G	A	G	A	G
8	8	8	8	8	8	8	8

2da FASE

G	C	A	G	A	G	A	G	← Sufijo más pequeño						
G	C	A	G	A	G	A	G							
							G	C	A	G	A	G	A	G
7	7	7	7	7	7	7	8	7 caracteres						

3era FASE

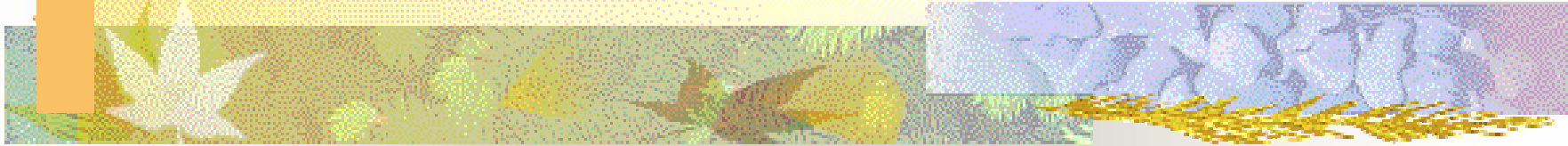
$$bmGs[m-1] = bmGs[m-1] - (m-1) = 8 - 7 = 1 \leftarrow \text{Calcula la última posición del patrón}$$

G	C	A	G	A	G	A	G
7	7	7	7	7	7	7	1

G	C	A	G	A	G	A	G												
				G	C	A	G	A	G	A	G	A	G						
7	7	7	7	7	4	7	1			4 caracteres									

G	C	A	G	A	G	A	G		
		G	C	A	G	A	G	A	G
7	7	7	2	7	4	7	1	2 caracteres	

Algoritmo Boyer-Moore-Horspool





Principales Características

- Es una simplificación del algoritmo de Boyer-Moore.
- Es fácil de implementar.
- Existe un preprocesamiento del patrón.
- Necesita $O(\sigma)$ en espacio y $O(m+\sigma)$ en tiempo (por el preprocesamiento).
- Realiza saltos determinados en el preprocesamiento.
- Compara de derecha a izquierda.
- Realiza la búsqueda del patrón en un tiempo $O(mn)$.
- Realiza un número promedio de comparaciones para un carácter entre $1/\sigma$ y $2/(\sigma+1)$.



Descripción

- Es considerado el mejor algoritmo de búsqueda de un patrón en un texto en aplicaciones usuales.
- El algoritmo escanea los caracteres del patrón con el texto iniciando con el carácter más a la derecha.
- En caso de una no ocurrencia o una ocurrencia total del patrón se usa una función preprocesada para saltar:
 - Salto del **mal carácter** (*bad-character shift*) (o salto de ocurrencia).
- Horspool propuso utilizar solamente el salto del mal carácter para calcular los saltos en el algoritmo de Boyer-Moore.



Descripción (cont.)

- El salto del mal carácter consiste en:
 - Alinear cada carácter del **alfabeto** Σ con la ocurrencia más a la derecha en $x[0, \dots, m-2]$.
 - Si el carácter no ocurre en el patrón x , la no ocurrencia de x puede incluir el carácter, y alinearlos en el lado más izquierdo del patrón.
- Esta operación (usada en el algoritmo BM) no es muy eficiente para alfabetos pequeños, pero cuando el alfabeto es grande comparado con la longitud del patrón llega a ser muy útil.
- Usarlo solo produce un algoritmo muy eficiente en la práctica.



Descripción (cont.)

- Se calcula el preprocesamiento del patrón de la siguiente forma:
 - Se calcula la distancia mínima entre el último carácter y la ocurrencia de cada carácter del alfabeto de la hilera principal.
- Para la búsqueda:
 - Consiste en la comparación de cada carácter del texto con las posiciones del patrón en el orden $m-1, 0, 1, 2, \dots$, y $m-2$, si se da una ocurrencia del patrón o no.
 - Cuando se encuentra una no ocurrencia, al hacer la primera comparación entre el patrón y el texto, el salto se calcula $bmBc[c]$, donde c es el carácter del texto.
 - Cuando se encuentra una no ocurrencia o una ocurrencia total, al hacer las siguientes comparaciones entre el patrón y el texto, el salto se calcula $bmBc[c]$, donde c es el carácter del patrón.



Código C

Ver [Algoritmo-BMH.txt](#)

Ejemplo

- Preprocesamiento:
 - Patrón = GCAGAGAG.

Otro Carácter	G	C	A	G	A	G	A	G
8	7	6	5	4	3	2	1	0

<i>c</i>	A	C	G	T
<i>bmBc</i> [<i>c</i>]	1	6	2	8

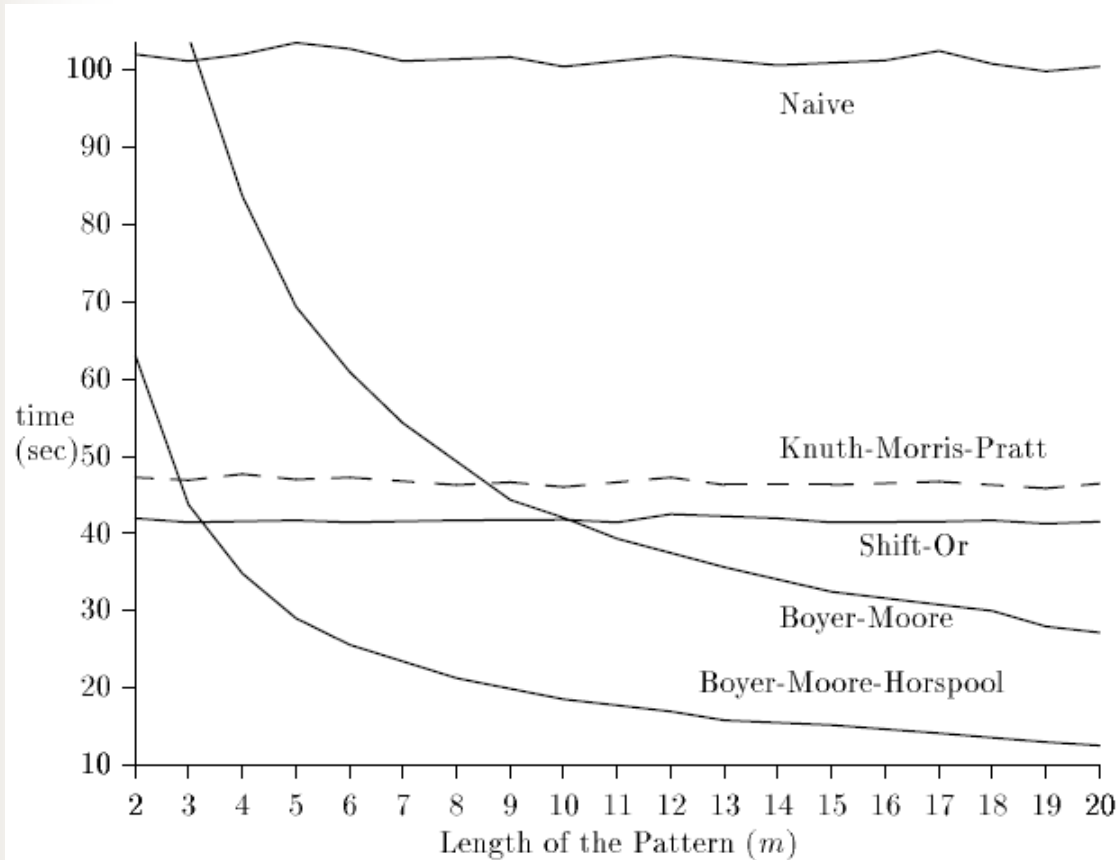
Ver [Algoritmo-BMH.pps](#)



Conclusiones

- El tiempo de ejecución para encontrar *matching* exacto entre un patrón x y un texto y , puede ser mejorado si se utiliza un algoritmo eficiente.
- Considerando la cantidad, cada vez mayor, de texto que se maneja, vale evaluar e ejecutar un algoritmo eficiente.
- En grandes colecciones de documentos en un dominio particular, el algoritmo Boyer-Moore-Horspool alcanza los mejores resultados al encontrar una secuencia en un texto.
- Aproximadamente, el algoritmo BMH realiza la búsqueda por lo menos dos veces más rápidamente que los otros algoritmos.

Conclusiones (cont.)



Resultados del experimento para buscar un conjunto de 1000 secuencias en un texto en Inglés. [BYGo92]



Referencias Bibliográficas

- Christian Charras y Thierry Lecroq. “Exact String Matching Algorithms”. Laboratorio de Informática de Rouen, Facultad de las Ciencias y las Técnicas, Universidad de Rouen. Francia, 1997. URL’s:
 - <http://www-igm.univ-mlv.fr/~lecroq/string/>.
 - <http://www-igm.univ-mlv.fr/~lecroq/string/string.pdf>.
- Ricardo Baeza-Yates y Gaston Gonnet. “A New Approach to Text Searching”. Communications of the ACM, 35(10):74-82. 1992.
- Presentación de Mauricio Ulate Quirós: “Algoritmos de Búsqueda Secuencial de Texto”, en el curso PF3394-Recuperación de Información. San José, Costa Rica, 2004.