

# TEMA 8. BÚSQUEDA SECUENCIAL

---

# Contenidos

- 1) Introducción
- 2) Algoritmo Trivial
- 3) Algoritmo Boyer-Moore
- 4) Búsqueda con error (aproximada)

# Bibliografía

## *Modern Information Retrieval:*

*Ricardo Baeza-Yates and Berthier Ribeiro-Neto*

*Addison Wesley, second edition **2011***

# 1. INTRODUCCIÓN

---

En algunos casos es necesario disponer de algoritmos que busquen una palabra o una secuencia de palabras en un documento.

Ejemplos:

- 1) Búsqueda en un documento que no está indexado.
- 2) El sistema de IR sólo indexa los documentos, pero no la posición.

También:

- Son muy usados en los programas de edición de textos o sistemas operativos (search, grep,...).
- Tienen muchas aplicaciones en bioinformática (búsqueda de patrones de ADN).
- Son muy útiles las búsquedas aproximadas, en las que se admiten errores de edición.

**Problema:** Encontrar la primera ocurrencia de un patrón (subcadena) en una cadena de caracteres.

Sea  $T[1..n]$  una cadena de caracteres de longitud  $n$  y  $P[1..m]$  el patrón buscado, de longitud  $m$ .

Diremos que el patrón  $P$  aparece en la posición  $s$  en el texto  $T$  si  $T[s..s+m-1]=P[1..m]$ .

### Ejemplo

$T = \text{aabbdaabc} \text{dad}$

$P = \text{aabc}$

$s = 6$

Vamos a enunciar el problema:

Se trata de encontrar la primera posición de izquierda a derecha en el texto  $T$  de una ocurrencia del patrón  $P$ .

## 2. ALGORITMO TRIVIAL

---

## Algoritmo trivial

- Deslizar una ventana de análisis sobre el texto incrementando cada vez un carácter.
- Comprobar en cada ventana si su secuencia de caracteres coincide con la del patrón P.
- Este análisis se hace de izquierda a derecha y en el momento que se encuentra un error se pasa a la siguiente ventana.

8

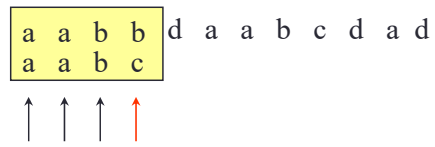
Vamos a dar como punto de partida el que llamamos algoritmo trivial. Consiste en hacer un barrido sistemático en el texto T, de izquierda a derecha y con incrementos de una posición. Se utiliza una ventana de comprobación de longitud m (la longitud del patrón), que se recorre de izquierda a derecha mientras los símbolos coincidan. En el momento que no sean símbolos iguales, la ventana corre una posición.



Ejemplo:

T= aabbdaabcdad

P= aabc



Veamos un ejemplo.

Como los caracteres 4 del texto y 4 del patrón son diferentes, la ventana corre una posición.

Ejemplo:

T= aabbdaabcdad

P= abc

a 

a	b	b	d
a	a	b	c

 a a b c d a d

↑ ↑

Como los caracteres 3 del texto y 2 del patrón son diferentes, la ventana corre una posición.

Ejemplo:

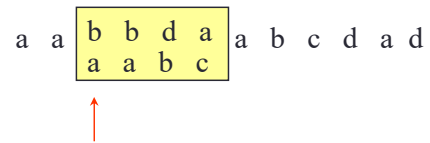
T= aabbdaabcdad

P= aabc

a a 

b	b	d	a
a	a	b	c

 a b c d a d



Como los caracteres 3 del texto y 1 del patrón son diferentes, la ventana corre una posición.

Ejemplo:

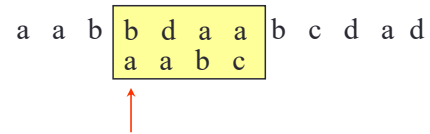
T= aabbdaabcdad

P= aabc

a a b 

b	d	a	a
a	a	b	c

 b c d a d



Como los caracteres 4 del texto y 1 del patrón son diferentes, la ventana corre una posición.

Ejemplo:

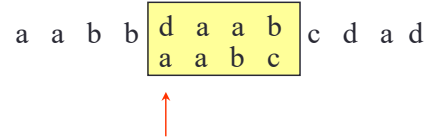
T= aabbdaabcdad

P= aabc

a a b b 

d	a	a	b
a	a	b	c

 c d a d



Como los caracteres 5 del texto y 1 del patrón son diferentes, la ventana corre una posición.

Ejemplo:

T= aabbdaabcdad

P= aabc

a	a	b	b	d	<table border="1"><tr><td>a</td><td>a</td><td>b</td><td>c</td></tr><tr><td>a</td><td>a</td><td>b</td><td>c</td></tr></table>	a	a	b	c	a	a	b	c	d	a	d
a	a	b	c													
a	a	b	c													
					↑	↑	↑	↑								

Coincidencia de todos los caracteres, la solución es 6.

```
def busqueda_trivial(T: str, P: str) -> int:
    for i in range(len(T) - len(P) + 1):
        j = 0
        while j < len(P) and T[i + j] == P[j]:
            j += 1
        if j == len(P):
            return i # encontrado
    return -1 # no encontrado
```

Coste temporal:  $O(|T| \times |P|)$

En la diapositiva se muestra el algoritmo trivial.  
En el peor caso la ventana corre  $n$  veces, y por cada posición de la ventana se hace la comprobación completa del patrón, de ahí el coste temporal que se indica.

### 3. ALGORITMO BOYER-MOORE

---



## Algoritmo de Boyer-Moore:

- Se almacena previamente en un vector indexado por los caracteres del alfabeto, la distancia a ***m*** de la última posición en que aparece cada carácter en el patrón.
- Se desliza una ventana de análisis sobre el texto, donde los saltos son determinados por los valores de este vector.
- La comparación del patrón con los caracteres de la ventana se realiza de derecha a izquierda.



Vamos a estudiar un algoritmo alternativo más eficiente. Se le conoce como el algoritmo de Boyer-Moore.

Antes de aplicar el algoritmo se calcula un vector de distancias indexado por los caracteres del alfabeto (texto y patrón) que ayudará a definir saltos en el proceso de recorrido de izquierda a derecha del texto T.

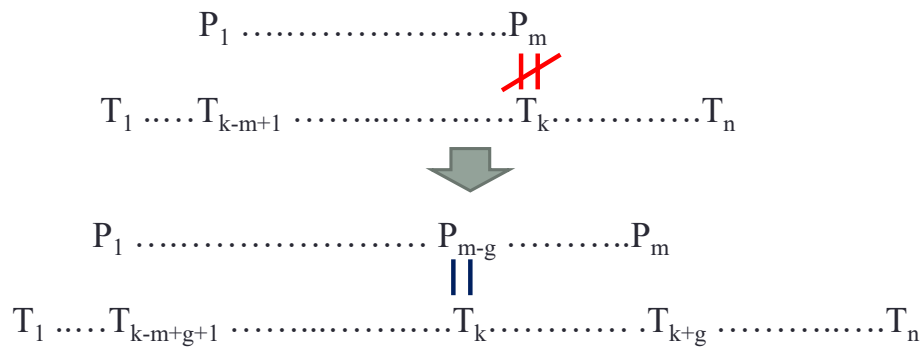
Con estos saltos se evitan comprobaciones.

Se desliza, al igual que en el caso del algoritmo trivial, una ventana de longitud m, la longitud del patrón, sobre el texto T, pero ahora cuando no hay coincidencia, el deslizamiento de la ventana sobre el texto no es necesariamente de 1 posición.

A diferencia del algoritmo trivial, una vez situada la ventana sobre el texto, la comprobación sobre los caracteres se efectúa de derecha a izquierda.

Tipos de desplazamiento:

- a) Si  $P_m$  no coincide con  $T_k$  entonces desplazamos el patrón de forma que alineamos  $T_k$  con la ocurrencia más a la derecha del símbolo  $T_k$  en P. Supongamos que  **$m-g$**  es la última posición en que aparece el símbolo  $T_k$  en P.



Dada una posición determinada de la ventana sobre el texto, supongamos que se compara el último símbolo del patrón  $P_m$  con el que le corresponde del texto, que ocupa la posición  $k$ , y no son iguales.

Consultamos en el vector de distancias la posición indexada por el símbolo  $T_k$ .

Nos dirá la ocurrencia de más a la derecha en el patrón del símbolo  $T_k$ .

Supongamos que es  $g$ , se corre la ventana hacia la derecha un salto de valor  $g$  de manera que se alinea  $T_k$  del texto con el símbolo  $P_{(m-g)}$  del patrón, que esta vez coincidirán.

Ejemplo:

↓  
P= abdbcdabbcb  
T= abcbbdabcbbcbabxcabcbbcababcb

Veamos un ejemplo.

Los símbolos de la posición 11 (tamaño del patrón) en el texto y en el patrón no coinciden.

El vector de distancias nos dirá que la ocurrencia más a la derecha del símbolo 'd' en el patrón está a una distancia de 5 del final del patrón.

Ejemplo:

↓  
P= abdbcdabbcb  
T= abcbbdabcbdcba**x**cabcbbcababcb

Por lo que desplazaremos la ventana 5 posiciones a la derecha, en lugar de hacer varios desplazamientos de 1, con las consiguientes comparaciones.

Hemos conseguido que ahora coincidan los símbolos de la posición 11 del texto y 6 del patrón.

Empieza otra vez la comparación de derecha a izquierda sobre la nueva ventana.

Como falla el primer símbolo, el x en este caso, y este símbolo no se encuentra en el patrón, en el vector de distancias se la habrá asignado una distancia m (11 en el ejemplo), y la ventana correrá m posiciones hacia la derecha.

Ejemplo:

P= abdbcdabbcb

T= abcbbdabcbdcba~~xcabcbbc~~ababcb

b) Supongamos que los últimos ***m-i*** caracteres del patrón coinciden con los últimos ***m-i*** caracteres de la cadena T, acabando en la posición ***k***.

$$P_{i+1} \dots \dots \dots P_m = T_{k-m+i+1} \dots \dots \dots T_k$$

Supongamos que  $P_i <> T_{k-m+i}$

En este caso hay dos posibilidades:

b1) Si la ocurrencia más a la derecha del carácter  $T_{k-m+i}$  en el patrón P es  $P_{i-g}$  entonces, como en el caso anterior, desplazamos el patrón ***g*** posiciones hacia la derecha, de modo que se alinea  $P_{i-g}$  con  $T_{k-m+i}$  y se comienza de nuevo a comparar  $P_m$  con  $T_{k+g}$ .

$$\begin{array}{ccccccc} P_1 & \dots & \dots & P_{i-g} & \dots & \dots & P_m \\ & & & \parallel & & & \\ T_1 & \dots & \dots & T_{k-m+g+1} & \dots & \dots & T_{k+g} \dots \dots \dots T_n \end{array}$$

Seguimos considerando otros casos.

Generalizamos para el caso en que dada una ventana determinada, los símbolos han coincidido de derecha a izquierda hasta la posición  $T_{(k-m+i+1)}$  en el texto y la posición  $P_{(i+1)}$  en el patrón.

Pero los siguientes símbolos comparados no coinciden, en este caso el símbolo  $T_{(k-m+i)}$  del texto y el  $P_{(i)}$  del patrón.

Nos encontramos con dos posibles situaciones, una en la que la información del vector de distancias es útil (b1) y otra en la que no (b2).

b1) Empezamos con el caso en el cual la información para el símbolo  $T_{(k-m+i)}$  en el vector de distancias es ***g***, entonces desplazamos la ventana ***g*** posiciones en el texto a partir de la posición del cursor,  $k-m+i$ , de forma que se alinea  $T_{(k-m+i)}$  con  $P_{(i-g)}$ , en lugar de con  $P_{(i)}$  como anteriormente.

b) Supongamos que los últimos ***m-i*** caracteres del patrón coinciden con los últimos ***m-i*** caracteres de la cadena T, acabando en la posición ***k***.

$$P_{i+1} \dots \dots \dots P_m = T_{k-m+i+1} \dots \dots \dots T_k$$

Supongamos que  $P_i <> T_{k-m+i}$

En este caso hay dos posibilidades:

b2) En el caso en que la ocurrencia más a la derecha de  $T_{k-m+i}$  en el patrón P esté más a la derecha de ***i*** entonces sólo se puede hacer un desplazamiento de un carácter.

b2) En el otro caso la información del vector de distancias no es útil, ya que la posición que indica está más a la derecha de ***i***, y no sirve para marcar el desplazamiento. En ese caso se desplaza la ventana 1 posición respecto de la anterior.

```

def boyer_moore(T: str, P: str) -> int:
    # cálculo de d
    d = dict((c, len(P) - P.rfind(c) - 1) for c in set(P))

    # búsqueda de P en T
    j = len(P) - 1
    while j < len(T):
        i = len(P) - 1
        while i >= 0 and P[i] == T[j]:
            i, j = i - 1, j - 1
        if i == -1:
            return j + 1 # encontrado
        elif len(P) - i > d.get(T[j], len(P)):
            j = j + len(P) - i
        else:
            j = j + d.get(T[j], len(P))
    return -1 # no encontrado

```

Coste: Mejor caso  $O(|T| / |P|)$   
Peor caso  $O(|T| \times |P|)$

Se calcula el vector de distancias  $d$  indexado por los caracteres del alfabeto, considerando tanto los del texto como los del patrón.

Para cada símbolo, se anota la distancia de la última ocurrencia por la derecha de ese símbolo en el patrón a la última posición del patrón,  $m$ . Para aquellos símbolos que no aparecen en el patrón se anota  $m$ .

En la diapositiva se muestra el algoritmo.



Ejemplo:

abracadabxdsaraxdabraabracadabra  
abraca<sup>d</sup>abra

a	b	c	d	r	x
0	2	6	4	1	11

Veamos un ejemplo.

El vector de distancias se muestra en la parte baja de la diapositiva.

Como el símbolo 'x' no aparece en el patrón, se le asigna el valor 11,

que se corresponde con la longitud del patrón.

Se coloca la ventana a la derecha del texto y la comparación empieza en la posición 11 de texto y patrón.

Son símbolos distintos, buscamos el símbolo 'd' en el vector y nos devuelve el valor 4, por lo que la ventana corre 4 posiciones a la derecha desde donde se encuentra el cursor en el texto.

Ejemplo:

abracadabxdsaraxdabraabracadabra  
abraca<sup>d</sup>abra  
abraca<sup>d</sup>abra<sup>a</sup>

a	b	c	d	r	x
0	2	6	4	1	11

Empieza la comparación entre las posiciones 15 del texto y 11 del patrón.

La comparación falla en las posiciones 13 del texto y 9 del patrón.

Buscamos el valor asignado al símbolo 'a' en el vector de distancias.

Como nos devuelve el valor 0 que queda a la derecha del valor de la posición actual, por ello se realiza por defecto un deslizamiento de la ventana a la derecha 1 posición.

Ejemplo:

abracadabxdsaraxdabraabracadabra  
abracadabra  
abracadabra  
abracadabra

a	b	c	d	r	x
0	2	6	4	1	11

Empieza la comparación entre las posiciones 16 del texto y 11 del patrón.

La comparación falla en las posiciones 16 del texto y 11 del patrón.

Buscamos el valor asignado al símbolo 'x' en el vector de distancias.

Como nos devuelve el valor 11, se realiza un deslizamiento de la ventana a la derecha 11 posiciones desde la posición del cursor en el texto.

Ejemplo:

Diagram illustrating the alignment of the word "abracadabra" with itself, showing the longest common subsequence (LCS) in blue and the characters not in the LCS in red. The LCS is "a b a c a b r a" and the characters not in the LCS are "x d a b r a".

a	b	c	d	r	x
0	2	6	4	1	11

Empieza la comparación entre las posiciones 27 del texto y 11 del patrón. La comparación falla en las posiciones 26 del texto y 10 del patrón. Buscamos el valor asignado al símbolo 'c' en el vector de distancias. Como nos devuelve el valor 6, se realiza un deslizamiento de la ventana a la derecha 6 posiciones desde donde se encuentra el cursor en el texto.

Ejemplo:

abracadabxdsaraxdabraabracadabra  
abracadabra  
abracadabra  
abracadabra  
abracadabra  
abracadabra

a	b	c	d	r	x
0	2	6	4	1	11

Esta vez el algoritmo acaba con una comparación con éxito completa.

El resultado es 22.

### EJERCICIO

Se pide indicar sobre la tabla, los desplazamientos que se realizarían en una búsqueda por Booyer-Moore del patrón **“acido acetico”** en la cadena **“se puede encontrar el acido acetico en”**.

s	e		p	u	e	d	e		e	n	c	o	n	t	r	a	r		e	l		a	c	i	d	o		a	c	e	t	i	c	o		e	n
a	c	i	d	o		a	c	e	t	i	c	o																									

Ejercicio

## Solución

s	e	p	u	e	d	e	e	n	c	o	n	t	r	a	r		e	l		a	c	i	d	o		a	c	e	t	i	c	o		e	n
a	c	i	d	o		a	c	e	t	i	c	o																							

a	c	d	e	i	l	n	o	p	r	s	t	
6	1	9	4	2	13	13	0	13	13	13	3	7

Solución del ejercicio:

En la parte inferior de la diapositiva se muestra el vector de distancias para este ejemplo.

Observamos que el blanco forma parte del vocabulario de símbolos, y que los símbolos 'l', 'n', 'p', 'r' y 's' tienen asignado el valor 13, longitud de patrón, por ser símbolos que no aparecen en el patrón.

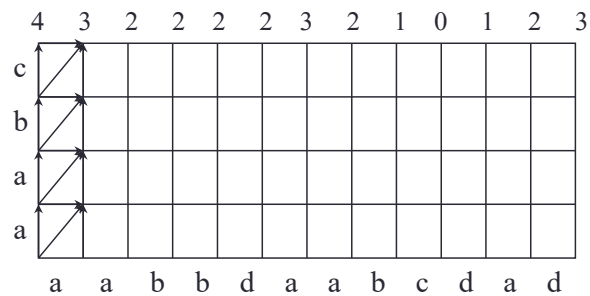
En la primera ventana la comparación falla en las posiciones 11 de texto y patrón, como el valor del vector de distancias para 'n' es 13, deslizamos la ventana 13 posiciones desde la posición actual en el texto, es decir, desde la posición 11.

## 4. BÚSQUEDA CON ERRORES (APROXIMADA)

---



- El algoritmo consiste en calcular múltiples distancias de Levenshtein entre el patrón y el texto. Para ello se relaja el punto de inicio de la comparación y se permite que el alineamiento empiece desde cualquier carácter del texto (y el inicial del patrón).
- No es necesario repetir el algoritmo de alineamiento para cada inicio, se puede hacer en una sola pasada, tal como muestra la figura.



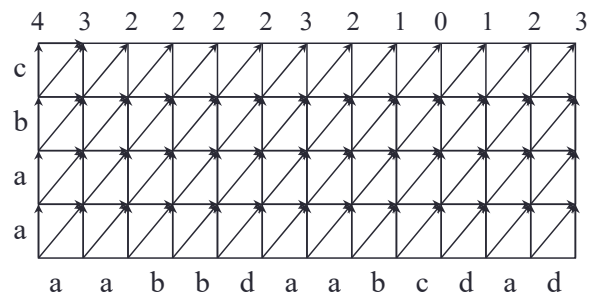
En la fila superior se encuentran las distancias de todos los posibles alineamientos de P con T.

Vamos a presentar otro algoritmo alternativo para encontrar el patrón en el texto. En este caso se proporcionan todas las ocurrencias del patrón en el texto con su coste acumulado.

Vamos a hacer una adaptación del algoritmo del cálculo de la distancia de Levenshtein relajando el punto de inicio de la comparación para poder encontrar segmentos en el texto que sean 'similares' al patrón.

En el eje horizontal de la tabla colocaremos el texto y en el vertical el patrón.

- El algoritmo consiste en calcular múltiples distancias de Levenshtein entre el patrón y el texto. Para ello se relaja el punto de inicio de la comparación y se permite que el alineamiento empiece desde cualquier carácter del texto (y el inicial del patrón).
- No es necesario repetir el algoritmo de alineamiento para cada inicio, se puede hacer en una sola pasada, tal como muestra la figura.



En la fila superior se encuentran las distancias de todos los posibles alineamientos de P con T.

Se completa la tabla con una inicialización diferente de la estudiada anteriormente. Cuando la tabla esta completa, en la fila superior tenemos la información de los errores acumulados para cada posición en el texto. A partir de esa información podemos encontrar los segmentos en el texto 'similares' al patrón con una cierta distancia.

```

def sust(a: str, b: str) -> int:
    return 1 if a != b else 0

def busqueda_PD(T: str, P: str):
    tabla = {}
    for j in range(len(P) + 1):
        tabla[j, 0] = j
    for i in range(len(T)):
        tabla[0, i + 1] = 0 # cualquier posición de inicio
        for j in range(len(P)):
            tabla[j + 1, i + 1] = min(
                tabla[j, i + 1] + 1,
                tabla[j + 1, i] + 1,
                tabla[j, i] + sust(T[i], P[j])
            )

```

Coste temporal:  $O(|P| \times |T|)$

El algoritmo se muestra en la diapositiva.

Se hace un barrido por columnas de izquierda a derecha, y en cada columna de abajo a arriba, como en el algoritmo ya estudiado. Sin embargo, para cada columna  $i$ , la fila 0 se inicializa al valor 0, para permitir que la comparación pueda empezar en cualquier columna (posición en el texto).

La versión del algoritmo que se muestra en la diapositiva está diseñada para completar la tabla aunque ya se haya encontrado alguna ocurrencia del patrón en el texto. Para encontrar las ocurrencias exactas del patrón en el texto se deben buscar valores de  $i$  tal que  $\text{tabla}[\text{len}(P), i] == 0$ , además, en ese caso  $i$  será la posición final del patrón sobre el texto.

Una vez calculada la matriz se puede utilizar para recuperar los diferentes segmentos en el texto con una 'similitud' al patrón por debajo de un umbral fijado,  $\text{tabla}[\text{len}(P), i] \leq \text{umbral}$ .

T= acdabpdqd

P= abcd

d	4	3	2	1	2	2	2	1	2	3
c	3	2	1	2	2	1	1	2	3	3
b	2	1	1	2	1	0	1	2	2	2
a	1	0	1	1	0	1	1	1	1	1
	0	0	0	0	0	0	0	0	0	0
	a	c	d	a	b	p	d	q	d	

Se ha considerado peso 1 para todas las operaciones de edición

Veamos un ejemplo: dado el texto T y el patrón P  
buscamos las posiciones  
en el texto del patrón con una distancia como  
mucho de 1.

Completamos la tabla y buscamos en la fila  
superior las posiciones con  
costes  $\leq 1$ .

A partir de cada una de estas posiciones se realiza  
un rastreo del camino para determinar el punto  
de inicio en el texto.

En el ejemplo, para una distancia máxima de 1  
encontramos la posiciones 3 y 7 en la fila superior.

En el rastreo hacia atrás siguiendo el mejor  
camino encontramos las soluciones:

Posiciones 1-3 segmento 'acd'

Posiciones 4-7 segmento 'abpd'

### EJERCICIO:

Se quiere buscar el patrón “estan” en el texto “estascasaseran”. Para ello se hace una búsqueda aproximada para obtener aquellos segmentos cuya distancia (de Levenshtein) al patrón es menor o igual que 1. Se pide completar la siguiente matriz que corresponde al algoritmo de búsqueda aproximada e indicar las soluciones, es decir, los segmentos del texto que son resultados de la búsqueda. Para el cálculo de la distancia se consideran pesos 1 para las Sustituciones, Inserciones y Borrados

n															
a															
t															
s															
e															
		e	s	t	a	s	c	a	s	a	s	e	r	a	n

Ejercicio

## Solución

<b>n</b>	5	4	3	2	1	1	2	3	3	3	3	4	4	3	2
<b>a</b>	4	3	2	1	0	1	2	2	3	2	3	3	3	2	3
<b>t</b>	3	2	1	0	1	2	2	3	2	2	2	2	2	2	3
<b>s</b>	2	1	0	1	2	1	2	2	1	2	1	1	1	2	2
<b>e</b>	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
		<b>e</b>	<b>s</b>	<b>t</b>	<b>a</b>	<b>s</b>	<b>c</b>	<b>a</b>	<b>s</b>	<b>a</b>	<b>s</b>	<b>e</b>	<b>r</b>	<b>a</b>	<b>n</b>

Segmentos encontrados con distancia 1:

- “esta” (posición 1-4) y
- “estas” (posición 1-5)

Solución del ejercicio