

# 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 necesarios disponer de algoritmos que busquen un 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

$T =$  aabbd<sup>aabc</sup>dad

$P =$  <sup>aabc</sup>

$s = 6$

## 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.

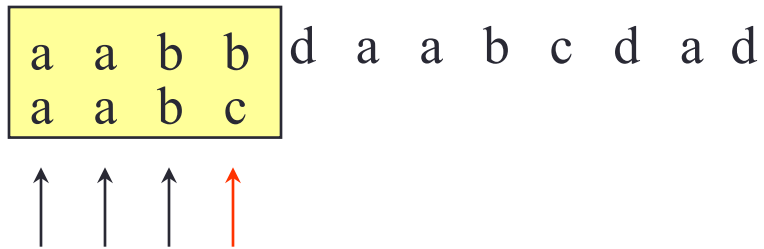




Ejemplo:

T= aabbdaabcdad

P= aabc

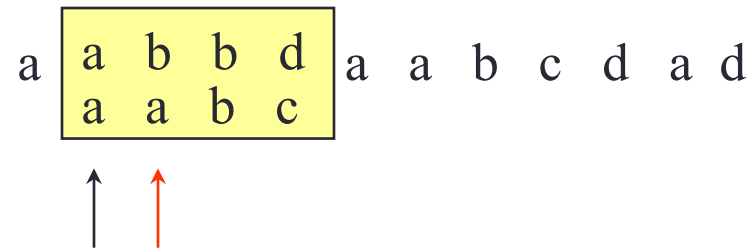




Ejemplo:

T= aabbdaabcbdad

P= aabc

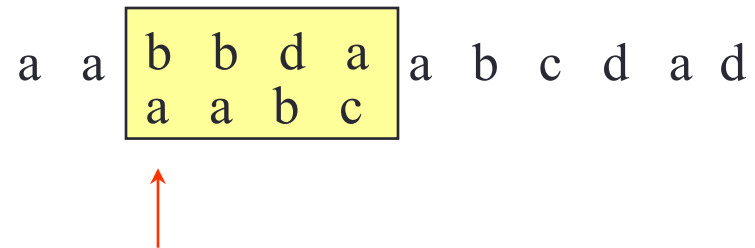




Ejemplo:

T= aabbdaabcdad

P= aabc





Ejemplo:


T= aabbdaabcdad

P= aabc

a a b 

b	d	a	a
a	a	b	c

 b c d a d





Ejemplo:

T= aabbdaabcdad

P= aabc

a a b b 

d	a	a	b
a	a	b	c

 c d a d

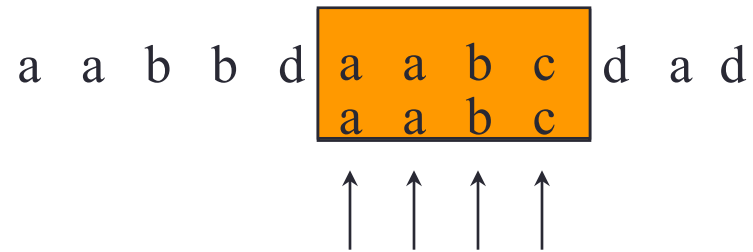
↑




Ejemplo:

T= aabbdaabcdad

P= aabc





```
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:  $O(|T| \times |P|)$

# 3. ALGORITMO BOYER-MOORE

---





# Algoritmo de Boyer-Moore:

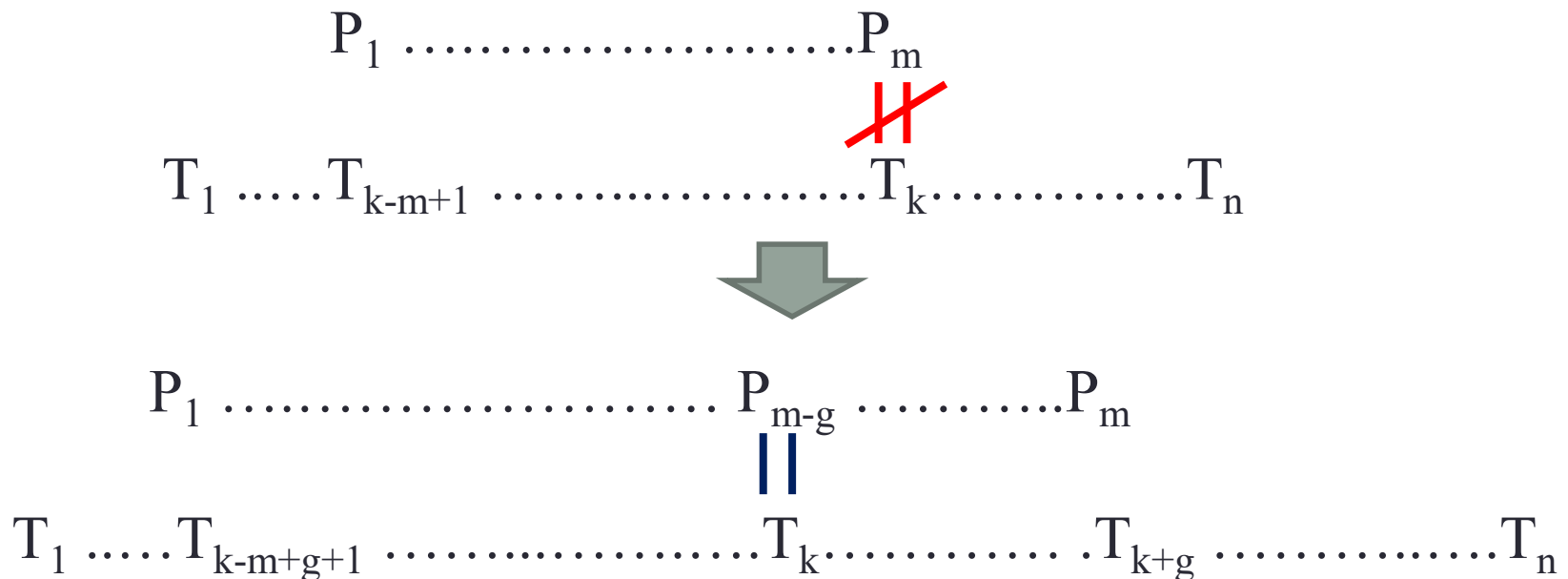
- Se almacena previamente en un vector indexado por los caracteres del alfabeto, la distancia a ***m*** de la última posición en que aparece cada carácter en el patrón.
- Se desliza una ventana de análisis sobre el texto, donde los saltos son determinados por los valores de este vector.
- La comparación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





## 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  $P$ .





Ejemplo:



P= abdbcdabbcb

T= abcbbdabcbbcbabxcabcbbcababcb



Ejemplo:



P=

abdbcdabbc**b**

T=

abcbbdabcb**d**cbabxcabcbbcababcb

Ejemplo:

P=

abdbcdabbcb

T= abcbbdabcbdcbabxcabcbbcababcb



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  $T_{k+g}$ .

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



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  $P_i \neq T_{k-m+i}$

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.



```
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  $O(|T| \times |P|)$





Ejemplo:

abracadabx**d**saraxdabraabracadabra  
abracad**d**abra

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



Ejemplo:

abracadabxdsaraxdabraabracadabra  
abracadabra  
abracadabra

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



Ejemplo:

abracadabxdsaraxdabraabracadabra  
abracadabra  
abracadabra  
abracadabra

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



Ejemplo:

abracadabxdsaraxdabraabracadabra  
abracadabra  
abracadabra  
abracadabra  
abracadabra

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



Ejemplo:

abracadabxdsaraxdabraabracadabra  
abracadabra  
abracadabra  
abracadabra  
abracadabra  
abracadabra  
abracadabra

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





## 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	i	d	o		a	c	e	t	i	c	o															
												a	c	i	d	o		a	c	e	t	i	c	o														
													a	c	i	d	o		a	c	e	t	i	c	o													
														a	c	i	d	o		a	c	e	t	i	c	o												
															a	c	i	d	o		a	c	e	t	i	c	o											
																						a	c	i	d	o			a	c	e	t	i	c	o			

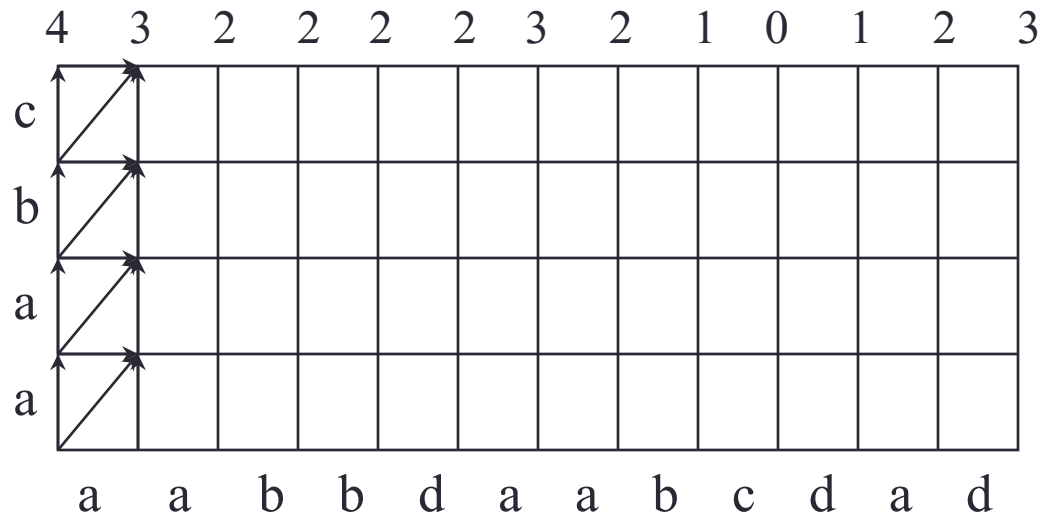
a	c	d	e	i	l	n	o	p	r	s	t	
6	1	9	4	2	13	13	0	13	13	13	3	7

## 4. BÚSQUEDA CON ERRORES (APROXIMADA)

---

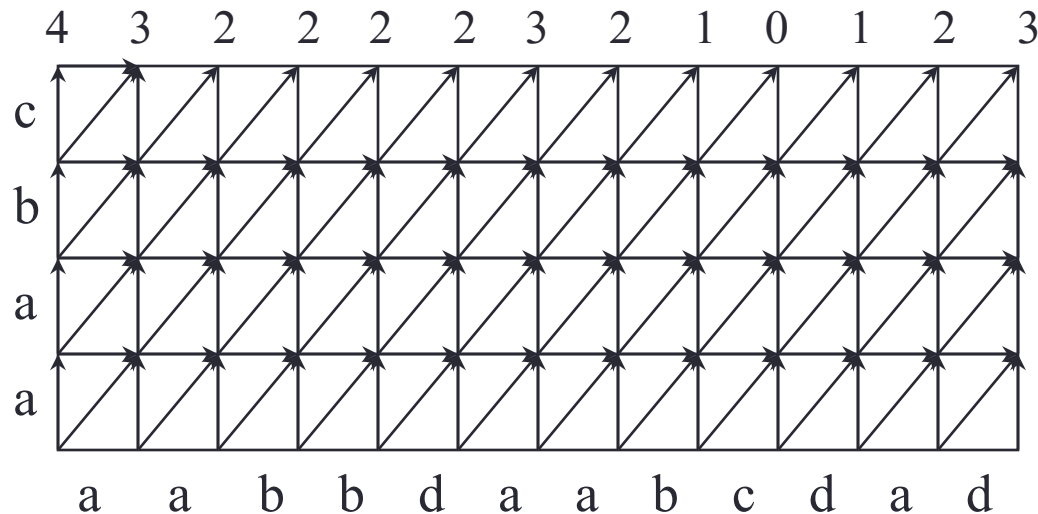


- El algoritmo consiste en calcular múltiples distancias de Levenshtein entre el patrón y el texto. Para ello se relaja el punto de inicio de la comparación y se permite que el alineamiento empiece desde cualquier carácter del texto (y el inicial del patrón).
- No es necesario repetir el algoritmo de 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.

- 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.



```
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:  $O(|P| \times |T|)$



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



## 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



## 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)