Búsqueda de Patrones

(Texto)

Definición

Búsqueda de patrones.

Búsqueda de elementos (texto, imágenes, señales…) que coincidan con un patrón determinado (una estructura básica).

!=

Reconocimiento de patrones.

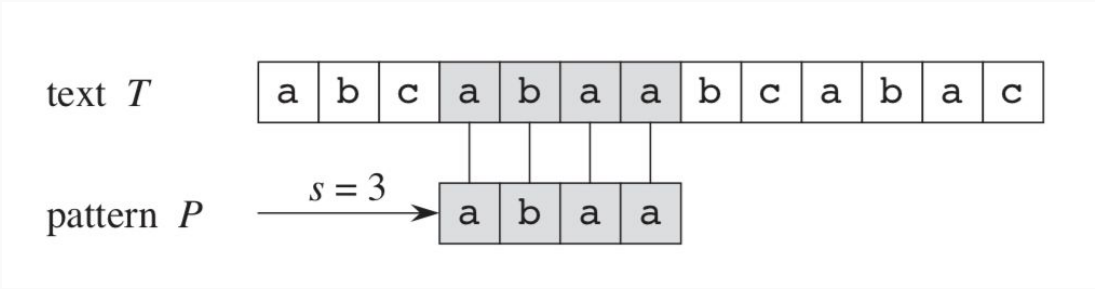
Extrae información de un conjunto de elementos para obtener un patrón recurrente en el conjunto.

Los algoritmos de búsqueda de patrones de texto pretenden encontrar una o varias ocurrencias de una cadena de texto dentro de otra.

Definición. Búsqueda de Patrones de texto (String matching)

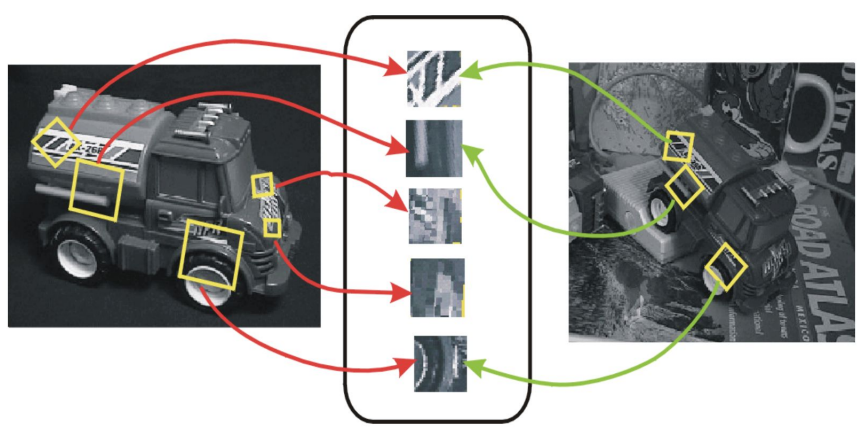
Formalmente, asumimos que:

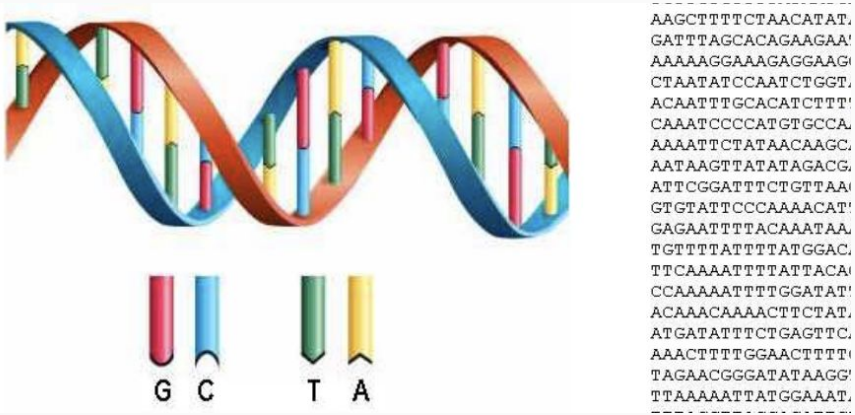
- El texto es un arreglo **`T[1..n]`** de tamaño **`n`**
- Un patrón es un arreglo **`P[1..m]`** de tamaño **`m`**, donde **`m<=n`**
- Los elementos de **`P`** y **`T`** son caracteres de un ***alfabeto finito***.
- Los elementos de **`P`** y **`T`** son conocidos como cadenas de caracteres (***strings***)



Aplicaciones

- Analizadores léxicos (parsers)
- Filtros de spam
- Procesadores de texto
- Librerías digitales
- grep (unix/linux)
- Motores de búsqueda (web)
- Procesamiento de lenguaje natural
- Biología molecular
- Procesamiento de imágenes y señales.



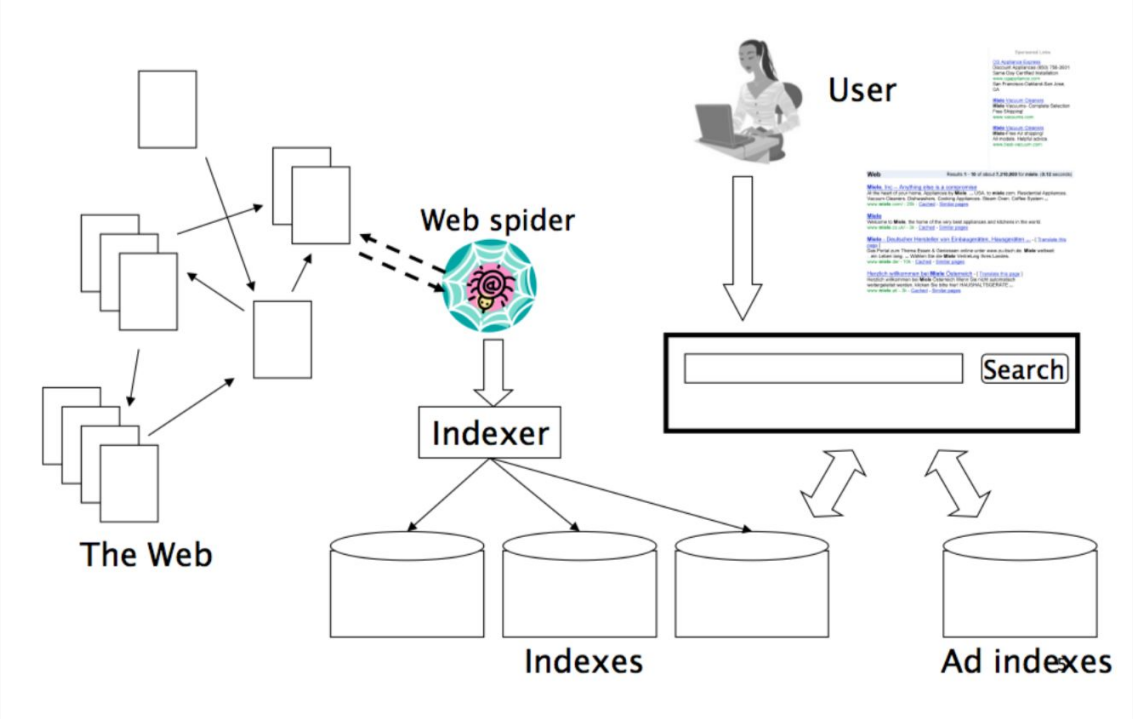


AXOLOTL
CORPUS PARALELO
NÁHUATL-ESPAÑOL

http://www.corpus.unam.mx/axolotl/

Aplicaciones

- Motores de búsqueda (web)



Algoritmos de Búsqueda de Patrones (de texto)

- Fuerza Bruta (Naive)
- Automatas Finitos
- Expresiones Regulares
- Knuth-Morris-Pratt
- Rabin-Karp
- ...

Algoritmos de Búsqueda de Patrones (de texto)

Complejidad:

Algorithm	Preprocessing time	Matching time
Naive	0	$O((n-m+1)m)$
Rabin-Karp	$\Theta(m)$	$O((n-m+1)m)$
Finite automaton	$O(m\lvert\Sigma\rvert)$	$\Theta(n)$
Knuth-Morris-Pratt	$\Theta(m)$	$\Theta(n)$

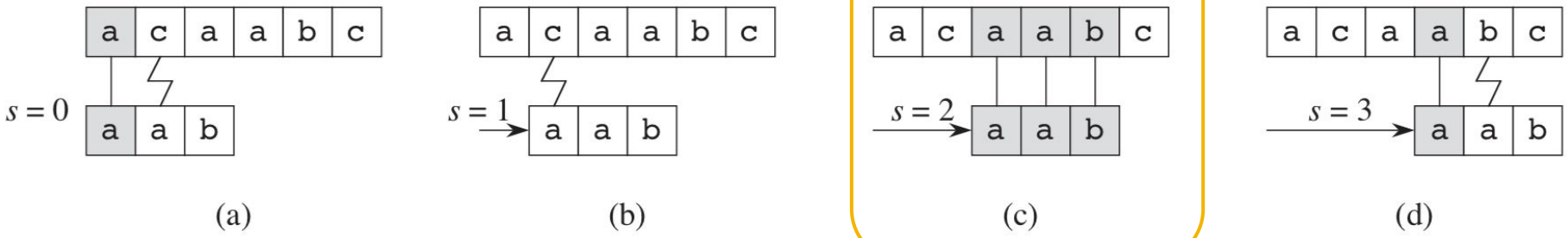
Fuerza Bruta

Compara el patrón con el texto una posición a la vez hasta que encuentra una discrepancia, se realiza mediante una iteración.

```
1  n = T.length
2  m = P.length
3  for s = 0 to n − m
4       if P[1..m] == T[s + 1..s + m]
5            print "Pattern occurs with shift" s
```

Fuerza Bruta

Compara el patrón con el texto una posición a la vez hasta que encuentra una discrepancia, se realiza mediante una iteración.

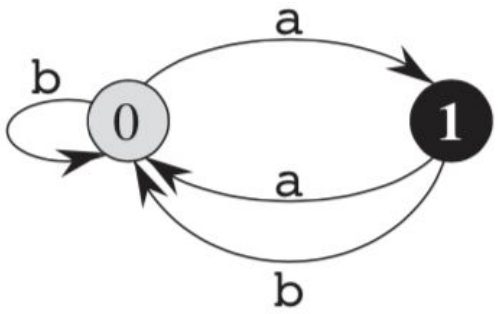


Autómatas Finitos

Un Autómata está formado por un **conjunto de estados**, incluyendo un **estado inicial** y **final**.

Recibe como **entrada** una **palabra** (una concatenación de símbolos del **alfabeto** del autómata) y dependiendo de ésta, la máquina puede cambiar de estados.

state	input a	input b
0	1	0
1	0	0

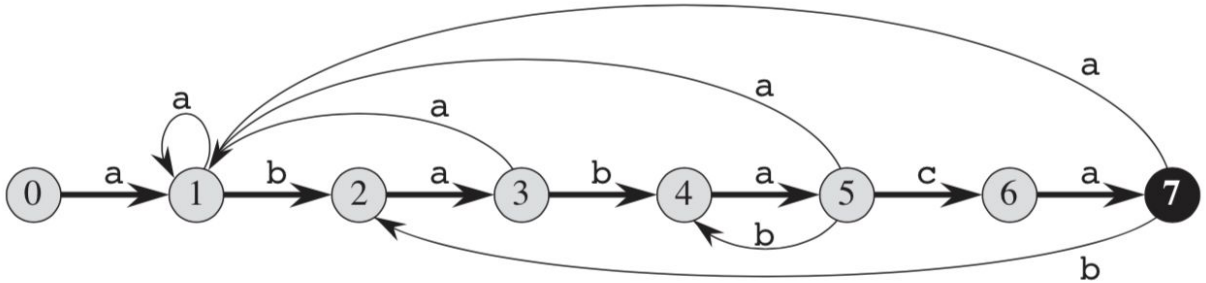


Estados = {0,1}

Alfabeto = {a,b}

Autómatas Finitos

Ejemplo.



state	input a	b	c	P
0	1	0	0	a
1	1	2	0	b
2	3	0	0	a
3	1	4	0	b
4	5	0	0	a
5	1	4	6	c
6	7	0	0	a
7	1	2	0	

i	—	1	2	3	4	5	6	7	8	9	10	11
$T[i]$	—	a	b	a	b	a	b	a	c	a	b	a
state $\phi(T_i)$	0	1	2	3	4	5	4	5	6	7	2	3

Patron = {a,b,a,b,a,c,a}

Expresiones Regulares (regex)

- Son cadenas de caracteres que **describen un patrón** para identificar un conjunto de cadenas (strings).
- Estos patrones se utilizan para identificar, reemplazar o dividir cadenas.
- ***P.ej.***
  - *Validar texto que introduce un usuario.*
  - *Verificar si el texto es un número telefónico, texto en mayúsculas, CURP, etc.*

Números pares: Terminan en dígitos 0,2,4,6,8

Expresión Regular: **[\\d]*[02468]** o **[1-9]*[02468]**

123 -> Falso, 12 -> Verdadero 3453458 -> Verdadero 1 -> Falso

Expresiones Regulares (regex). Sintaxis.

Una expresión regular está compuesta de caracteres literales y símbolos especiales.

Regular Expression	*Matches*	*Example*
x	a specified character x	Java matches Java
.	any single character	Java matches J..a
(ab|cd)	ab or cd	ten matches t(en|im)
[abc]	a, b, or c	Java matches Ja[uvwx]a
[^abc]	any character except a, b, or c	Java matches Ja[^ars]a
[a-z]	a through z	Java matches [A-M]av[a-d]
[^a-z]	any character except a through z	Java matches Jav[^b-d]
[a-e[m-p]]	a through e or m through p	Java matches [A-G[I-M]]av[a-d]
[a-e&&[c-p]]	intersection of a-e with c-p	Java matches [A-P&&[I-M]]av[a-d]

Regular Expression	*Matches*	*Example*
\d	a digit, same as [0-9]	Java2 matches "Java[\\d]"
\D	a non-digit	$Java matches "[\\D][\\D]ava"
\w	a word character	Java1 matches "[\\w]ava[\\w]"
\W	a non-word character	$Java matches "[\\W][\\w]ava"
\s	a whitespace character	"Java 2" matches "Java\\s2"
\S	a non-whitespace char	Java matches "[\\S]ava"
*p**	zero or more occurrences of pattern *p*	aaaabb matches "a*bb" ababab matches "(ab)*"
p+	one or more occurrences of pattern *p*	a matches "a+b*" able matches "(ab)+.*"
p?	zero or one occurrence of pattern *p*	Java matches "J?Java" Java matches "J?ava"
p{n}	exactly n occurrences of pattern *p*	Java matches "Ja{1}.*" Java does not match ".{2}"
p{n,}	at least n occurrences of pattern *p*	aaaa matches "a{1,}" a does not match "a{2,}"
p{n,m}	between n and m occurrences (inclusive)	aaaa matches "a{1,9}" abb does not match "a{2,9}bb"

Expresiones Regulares (regex). Sintaxis.

- Caracteres: Un caracter puede ser una **letra**, **digito** o **_**

  `\w` es lo mismo que **`[a-z[A-Z][0-9]_]`** **o** **`[a-zA-Z0-9_]`**

- Cuantificadores:
  - **`A*`** indica cero o mas A's
  - **`A+`** indica una o mas A's
  - **`A?`** Indica 0 o una A.
  - **`A{3}`** indica AAA exactamente
  - **`A{3,6}`** indica de 3 a 6 A's

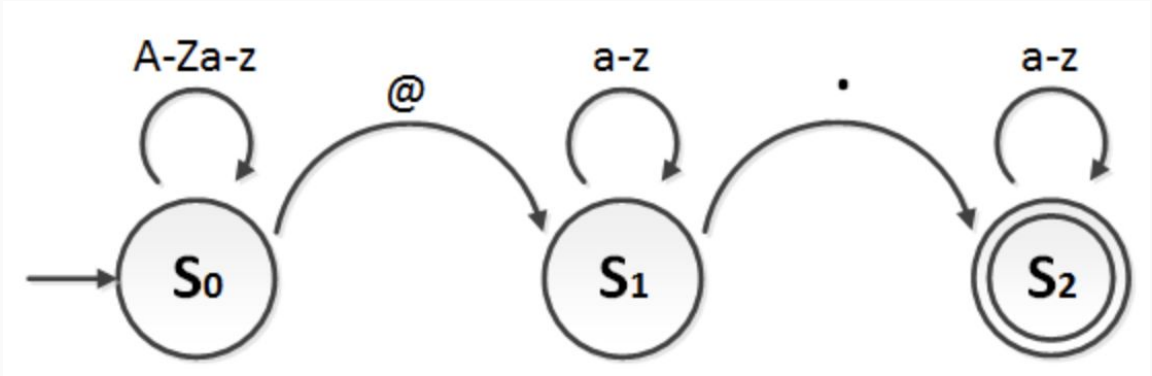
- Parentesis: Utilizado para agrupar patrones:
  - **`(ab){3}`** **indica** **`ababab`**
  - **`ab{3}`** **indica** **`abbb.`**

Expresiones Regulares (regex)

- Validación de correo electrónico.

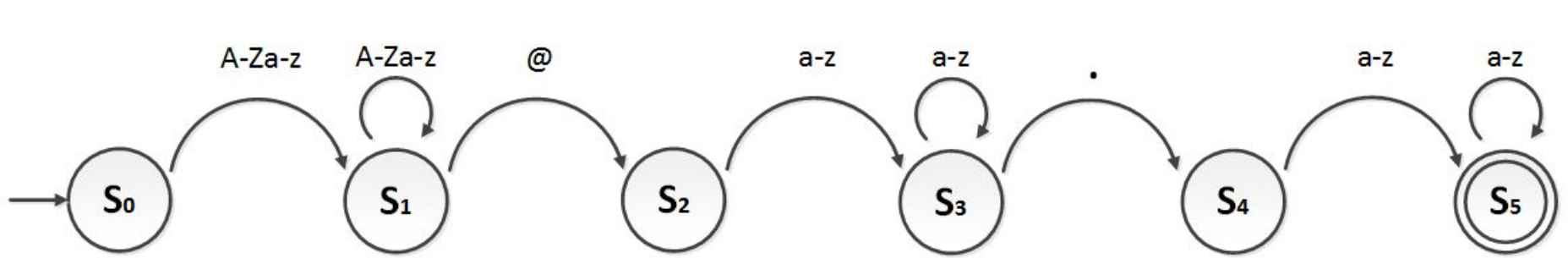
nombre@correo.com

texto @ texto . texto



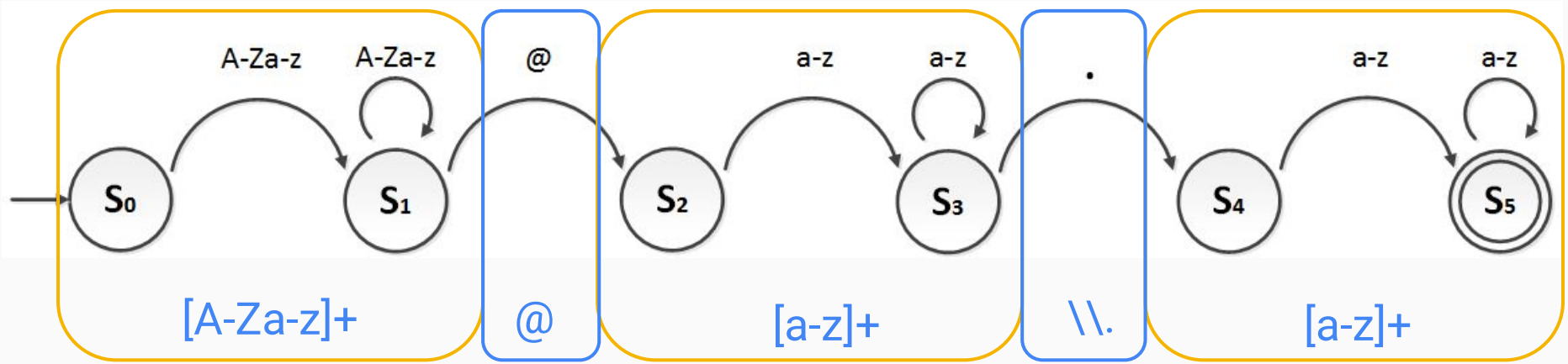
Expresiones Regulares (regex)

- Extendiendo las validaciones...



Expresiones Regulares (regex)

- Creando expresión regular...



Knuth-Morris-Pratt (KMP)

Se diferencia del algoritmo de fuerza bruta al llevar un registro de la información obtenida de las comparaciones previas.

Utiliza una función que lleva el registro de las fallas de comparaciones previas para reutilizarlas y evitar comparar cada caracter del texto nuevamente.

```
1   n = T.length
2   m = P.length
3   π = COMPUTE-PREFIX-FUNCTION(P)
4   q = 0                                  // number of characters matched
5   for i = 1 to n                         // scan the text from left to right
6       while q > 0 and P[q + 1] ≠ T[i]
7           q = π[q]                       // next character does not match
8       if P[q + 1] == T[i]
9           q = q + 1                      // next character matches
10      if q == m                          // is all of P matched?
11          print "Pattern occurs with shift" i − m
12          q = π[q]                       // look for the next match
```

```
COMPUTE-PREFIX-FUNCTION(P)
1   m = P.length
2   let π[1 . . m] be a new array
3   π[1] = 0
4   k = 0
5   for q = 2 to m
6       while k > 0 and P[k + 1] ≠ P[q]
7           k = π[k]
8       if P[k + 1] == P[q]
9           k = k + 1
10      π[q] = k
11  return π
```

Rabin-Karp

Calcula un hash para el patrón y para cada subsecuencia de caracteres del texto a comparar. Si los valores hash son distintos, el algoritmo calcula el hash para la siguiente secuencia. Si los valores son iguales, se realiza una comparación de fuerza bruta entre el patrón y la secuencia. De esta manera solo hay una comparación por subsecuencia y la fuerza bruta se utiliza solo cuando los valores hash concuerdan.

```
RABIN-KARP-MATCHER(T, P, d, q)
 1  n = T.length
 2  m = P.length
 3  h = d^(m−1) mod q
 4  p = 0
 5  t_0 = 0
 6  for i = 1 to m                          // preprocessing
 7      p = (dp + P[i]) mod q
 8      t_0 = (dt_0 + T[i]) mod q
 9  for s = 0 to n − m                      // matching
10      if p == t_s
11          if P[1..m] == T[s + 1..s + m]
12              print "Pattern occurs with shift" s
13      if s < n − m
14          t_{s+1} = (d(t_s − T[s + 1]h) + T[s + m + 1]) mod q
```