

Grado en Ingeniería Informática

Computabilidad y Algoritmia

PRÁCTICA 3

Expresiones regulares

Factor de ponderación: 6

Semana del 29 de septiembre al 3 de octubre 2025

1. Objetivos

En el ámbito de la teoría de autómatas y los lenguajes formales, una expresión regular (*regular expression* o *regex*) es un mecanismo formal que permite describir lenguajes regulares. Es decir, toda expresión regular representa a un lenguaje regular [1]. Desde un punto de vista más general podríamos decir que una expresión regular es una secuencia de caracteres que conforma un patrón de búsqueda o de reconocimiento (*“matching”*). Estas secuencias de caracteres, que se describen mediante una sintaxis especial, proporcionan una manera muy flexible y eficiente de buscar o reconocer cadenas en un texto.

Puesto que las expresiones regulares son una poderosa herramienta utilizada en la programación y el procesamiento de texto para buscar, analizar y manipular patrones dentro de cadenas de caracteres, los principales lenguajes de programación tienen soporte para la utilización de expresiones regulares (C++, Python, Java, Ruby, etc.). Esto se debe a que habitualmente los programadores necesitan trabajar con cadenas de texto (strings) para validar los datos ingresados por el usuario, validar formatos de una URL, de un correo electrónico, reemplazar palabras en párrafos, etc.

A nivel más general, podemos decir que las expresiones regulares se utilizan en una amplia variedad de contextos en la informática y la programación. Algunos de estos ejemplos se enumeran a continuación:

- **Búsqueda, validación y formateo de datos:** se emplean para buscar información específica en textos, validar o formatear datos de entrada (como direcciones de correo electrónico o números de teléfono), y extraer datos de interés de documentos o archivos.
- **Procesamiento de texto:** las expresiones regulares son fundamentales en el procesamiento de texto, desde la manipulación de cadenas en lenguajes de programación hasta la búsqueda y reemplazo en editores de texto avanzados.
- **Análisis de logs y texto:** en aplicaciones de registro (log) y análisis de texto, las expresiones regulares ayudan a identificar patrones (principalmente errores o eventos de interés) en grandes volúmenes de datos.
- **Web scraping:** se utilizan para extraer información específica de páginas web al buscar y analizar el código fuente HTML.
- **Lenguajes de consulta:** en bases de datos y motores de búsqueda, se utilizan expresiones regulares para realizar consultas avanzadas que coincidan con patrones de datos específicos.

En resumen, las expresiones regulares son una herramienta esencial en el mundo de la programación y el procesamiento de texto, permitiendo a los desarrolladores y analistas trabajar de manera eficiente con patrones de datos en una amplia gama de aplicaciones. Su versatilidad y potencia las convierten en una habilidad valiosa para cualquier persona que trabaje con datos textuales en entornos informáticos. Es por ello que el objetivo de la práctica es profundizar en la definición y uso de expresiones regulares, desde una notación teórica básica hasta la utilización de operadores extendidos.

IMPORTANTE: el entregable de esta práctica será un informe en formato .pdf que incluya los ejercicios planteados en el guión así como aquellos que se puedan plantear durante la propia sesión práctica.

2. Operadores básicos

Una expresión regular es una cadena de caracteres que define un patrón de búsqueda. Estos patrones pueden incluir caracteres literales, como letras o números, así como metacaracteres que representan clases de caracteres, repeticiones, alternancias y más. Las expresiones regulares permiten especificar de manera flexible y concisa cómo debe lucir un fragmento de texto que deseamos encontrar o manipular. Para poder especificar el patrón de búsqueda se utiliza una sintaxis de expresiones regulares que, además de incluir los operadores básicos de disyunción, concatenación y repetición, admite extensiones de dichos operadores.

Como primera aproximación vamos a realizar una serie de ejercicios en los que trabajaremos únicamente con los operadores básicos:

- **Disyunción:** el operador de disyunción se representa con el símbolo $|$ y se utiliza para crear una alternativa entre diferentes elementos. Permite buscar una de varias opciones. Por ejemplo, la expresión regular $a|b$ coincidirá con la cadena “a” o con la cadena “b”, ya que la disyunción implica una elección entre ambos elementos.
- **Concatenación:** la concatenación es el proceso de unir elementos o caracteres de manera secuencial. En una expresión regular, la concatenación se representa generalmente escribiendo los caracteres o patrones uno tras otro, aunque en algunos contextos se utiliza el operador \cdot .

Por ejemplo, la expresión regular abc permitiría buscar la secuencia “abc” en una cadena. Esto es, una “a” seguida de “b” seguida de “c”.

- **Repetición:** los operadores de repetición $*$ y $+$ permiten especificar cuántas veces se debe repetir un elemento o patrón en una cadena.
 - $*$: cero o más repeticiones de la expresión. Por ejemplo, la expresión regular a^*b coincidirá con cadenas de cero o más “a’s” que vengan seguidas de una única “b”.
 - $+$: una o más repeticiones del elemento anterior. Por ejemplo, la expresión regular ab^+a coincidirá con cadenas que tengan una “a” seguida de una o más “b’s” seguidas de una última “a”.

Hay que tener en cuenta que la precedencia (de mayor a menor) de los operadores anteriores es: repetición, concatenación y, finalmente, disyunción. Para modificar el orden en el que se aplican dichos operadores es necesario utilizar los paréntesis. Por ejemplo, véase la diferencia entre las expresiones regulares siguientes:

- $a|b^*$: casa con la cadena “a” o bien con cualquier cadena de cero o más “b’s”.
- $(a|b)^*$: casa con cualquier cadena de cero o más “a’s” o “b’s”. En este caso, tanto las “a’s” como las “b’s” podrían venir en cualquier orden y número.

Para poder comprender mejor las expresiones regulares anteriores y la forma en la que se producen las coincidencias, utilizaremos la herramienta RegExper [2]. Esta herramienta nos ofrece una interfaz para visualizar gráficamente una expresión regular a través de un diagrama. A continuación se incluyen dos imágenes que muestran el uso de esta herramienta.

En la primera imagen podemos ver el diagrama asociado a la expresión regular $a|b^*$. Como vemos el camino en el que se encuentra la “a” es un camino totalmente independiente del camino en el que se encuentra la “b”.

En la segunda de las imágenes podemos ver el diagrama asociado a la expresión regular $(a|b)^*$. En este caso, es fácil percibir cómo las repeticiones se aplican a cualquiera de las opciones “a” o “b”.

REGEXPER
You thought you only had two problems...

Changelog
Documentation
Source on GitLab

`a|b*`

Display Download SVG // Download PNG // Permalink

REGEXPER
You thought you only had two problems...

Changelog
Documentation
Source on GitLab

`(a|b)*`

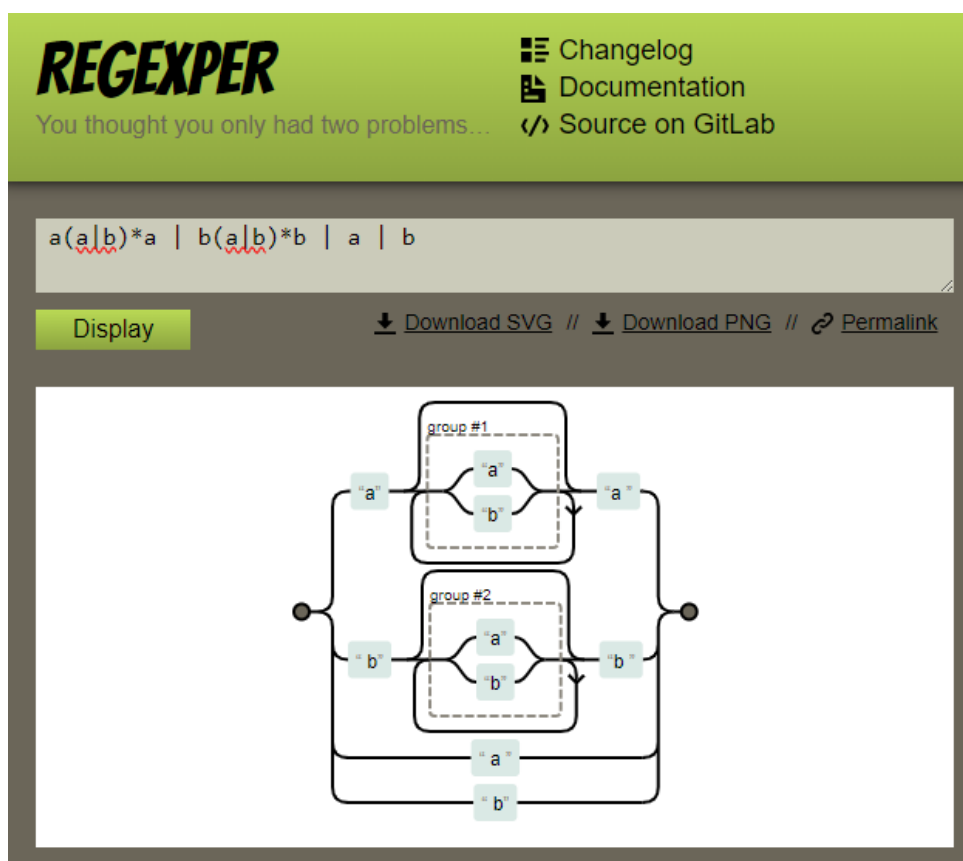
Display Download SVG // Download PNG // Permalink

Otro ejemplo algo más complejo se centrará en la siguiente expresión regular:

$$a(a|b)^*a \mid b(a|b)^*b \mid a \mid b$$

La expresión regular anterior representa el lenguaje formado por todas las cadenas de “a’s” y “b’s” en las que la cadena empieza y termina por el mismo símbolo.

En el siguiente diagrama se pueden apreciar las cuatro opciones o caminos posibles: que la cadena empiece y termine por “a”, que empiece y termine por “b”, que la cadena sea “a” directamente o que la cadena sea “b”.



2.1. Ejercicios prácticos

Teniendo en cuenta los operadores básicos [1] y la herramienta de visualización anterior [2], diseñe expresiones regulares que representen a los siguientes lenguajes regulares:

1. Cadenas sobre el alfabeto $\{a, b\}$ con longitud impar.
2. Cadenas sobre el alfabeto $\{a, b\}$ con longitud igual a 5.
3. Cadenas sobre el alfabeto $\{a, b, c\}$ con una "a" en la antepenúltima posición.
4. Cadenas sobre el alfabeto $\{a, b\}$ con número de "a's" par o número de "b's" impar.
5. Cadenas w sobre el alfabeto $\{0, 1\}$ tales que $2 \leq |w| \leq 5$
6. Cadenas sobre el alfabeto $\{0, 1\}$ con longitud múltiplo de 3.
7. Cadenas sobre el alfabeto $\{0, 1\}$ con una longitud que no sea múltiplo de 3.
8. Cadenas w sobre el alfabeto $\{0, 1\}$ tal que $w = 0^n 1^m$ con $n + m$ impar.
9. Cadenas sobre el alfabeto $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$ que tengan como máximo dos ceros.
10. Cadenas sobre el alfabeto $\{x, y, z\}$ que no contengan dos símbolos x consecutivos.

En el informe de la práctica, se incluirá cada uno de los lenguajes anteriores, junto con la expresión regular que lo representa, el diagrama de representación visual de la expresión [2], así como un ejemplo de 5 cadenas pertenecientes al lenguaje y 5 cadenas que no pertenezcan al lenguaje en cuestión.

3. Operadores extendidos

Para aportar mayor potencia y flexibilidad a las expresiones regulares, en la mayor parte de los lenguajes de programación existe un conjunto de operadores extendidos que, generalmente, están disponibles a través de las correspondientes librerías para expresiones regulares (por ejemplo, `regex` en el caso de C++). A continuación se relacionan algunos de los principales operadores extendidos que se suelen utilizar para definir expresiones regulares:

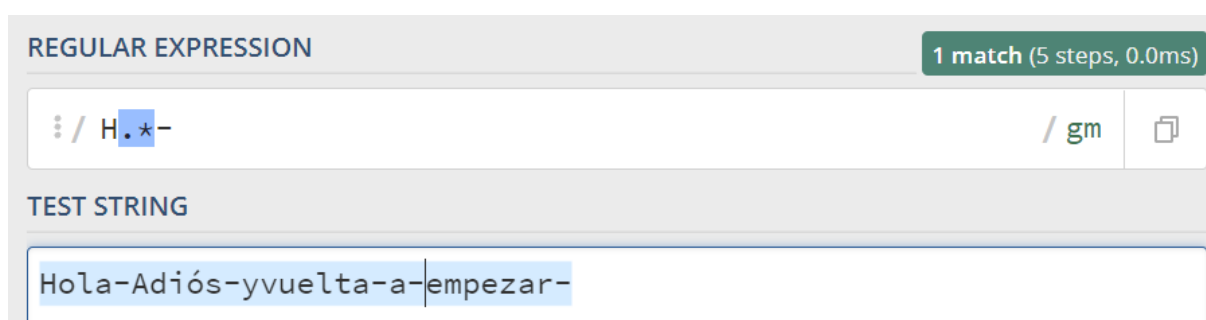
<code>[A – Z]</code>	Cualquier letra mayúscula
<code>[0 – 9]</code>	Cualquier dígito
<code>[A – Za – za0 – 9]</code>	Cualquier letra o dígito
<code>.</code>	Cualquier caracter (excepto el salto de línea)
<code>+</code>	Una o más ocurrencias
<code>*</code>	Cero o más ocurrencias
<code>?</code>	Cero o una única ocurrencia
<code>{m}</code>	Exactamente <i>m</i> ocurrencias
<code>{m,n}</code>	Entre <i>m</i> y <i>n</i> ocurrencias (ambas incluidas)
<code> </code>	Una cosa u otra (or)
<code>^</code>	Primera ocurrencia (comienzo del patrón)
<code>\$</code>	Última ocurrencia (final del patrón)
<code>\+, *, \\\</code>	Los caracteres especiales hay que antecederlos de \
<code>\d</code>	Cualquier dígito
<code>\D</code>	Cualquier caracter que no sea un dígito
<code>\w</code>	Cualquier letra o dígito
<code>\W</code>	Cualquier caracter que no sea letra ni dígito
<code>\s</code>	Cualquier blanco
<code>\S</code>	Cualquier caracter que no sea un blanco
<code>()</code>	Captura de grupos y/o subgrupos

Para entender mejor y poner en práctica la sintaxis de las expresiones regulares y sus operadores extendidos comience por estudiar los siguientes tutoriales interactivos:

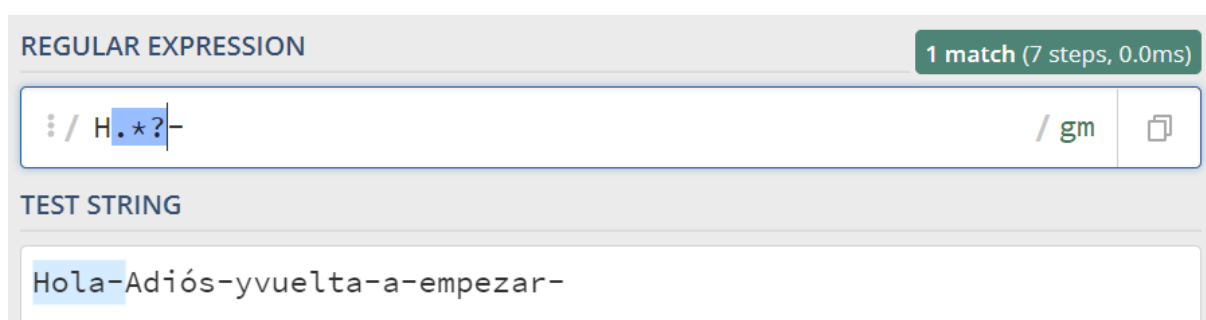
- [RegexOne: Learn Regular Expressions with simple, interactive exercises \[3\]](#).
- [RegexLearn: Learn Regular Expressions \[4\]](#).

En lo que se refiere a la sintaxis, es importante hacer inciso en dos aspectos: el comportamiento “*greedy*” del operador `*` y la utilización de los agrupamientos para luego acceder a partes concretas de la secuencia coincidente.

En primer lugar, es importante tener en cuenta que los cuantificadores de repetición por defecto tienen un comportamiento ávido (*greedy*). Para mostrar este comportamiento se ha utilizado una herramienta para el testeo online con expresiones regulares [5]. Si nos fijamos en la expresión regular siguiente (cadenas que empiezan por H seguida de cero o más caracteres y terminan en guión), veremos que, por defecto, el `.*` casará con toda la cadena, llegando hasta el último guión (-). A este comportamiento se le llama ávido (voraz) o *greedy* porque casa con la mayor cadena posible [6].



Sin embargo, si utilizáramos `.*?` estaríamos usando un cuantificador perezoso o *lazy*. De esta forma, al realizar el emparejamiento se pararía con la primera subcadena (subcadena más corta) que cumpla con el patrón especificado [7].



3.1. Ejercicios prácticos

Teniendo en cuenta los operadores extendidos utilizados en los tutoriales [3, 4] y disponibles en la herramienta interactiva [5], diseñe expresiones regulares que permitan reconocer o encontrar los siguientes patrones:

1. Direcciones de correos electrónicos de estudiantes de la Universidad de La Laguna.
2. Palabras que terminen por una vocal.
3. Números enteros.
4. Texto que se encuentre entre paréntesis.
5. Código postal en España.
6. Palabras que contienen solo letras mayúsculas.
7. Números de teléfono en formato `prefijo XXX-XXX-XXX`, donde el prefijo del país puede indicarse empezando por 00 o bien con un símbolo +; por ejemplo, 0034 o +34 para España.
8. Fecha en formato `DD/MM/AAAA`
9. Palabras de al menos 10 letras de longitud.
10. Palabras que terminen con *“ing”* o *“ed”*.

En el informe de la práctica, se incluirá la relación de expresiones regulares diseñadas en cada uno de los casos anteriores así como una captura de la herramienta Regex 101 [5] en la que se vea un texto con al menos 5 coincidencias del patrón.

Recuerda que la sintaxis de algunos operadores extendidos utilizados en expresiones regulares pueden variar en función del lenguaje de programación que estés utilizando, ya que algunos lenguajes tienen implementaciones ligeramente diferentes para expresiones regulares.

4. Criterios de evaluación

Se señalan a continuación los aspectos más relevantes (la lista no es exhaustiva) que se tendrán en cuenta a la hora de evaluar esta práctica:

- Se valorará que el alumnado haya realizado, con anterioridad a la sesión de prácticas, y de forma efectiva, todas las tareas propuestas en este guión. Esto incluye la realización de los tutoriales interactivos aquí propuestos.
- Se valorarán las soluciones ofrecidas a los ejercicios planteados en este guión, tanto

en relación a los operadores básicos como a los operadores extendidos.

- También se valorará la capacidad de ofrecer soluciones a los ejercicios que se planteen durante la propia sesión de prácticas.
- Se valorará la utilización de las herramientas presentadas en el guión de esta práctica.
- Finalmente, se valorará el formato y claridad del informe de prácticas presentado, así como la adecuación y cumplimiento de las instrucciones aquí proporcionadas.

Si el alumnado tiene dudas respecto a cualquiera de estos aspectos, debiera acudir al foro de discusiones de la asignatura para plantearlas allí. Se espera que, a través de ese foro, el alumnado intercambie experiencias y conocimientos, ayudándose mutuamente a resolver dichas dudas. También el profesorado de la asignatura intervendrá en las discusiones que pudieran suscitarse, si fuera necesario.

Referencias

- [1] Transparencias del Tema 2 de la asignatura: Autómatas finitos y lenguajes regulares, [https://campusvirtual.ull.es/2526/ingenieriaytecnologia/mod/resource/view.php?id=11856](https://campusvirtual ull.es/2526/ingenieriaytecnologia/mod/resource/view.php?id=11856)
- [2] RegExper: displaying regular expressions, <https://regexper.com>
- [3] RegexOne: Learn Regular Expressions with simple, interactive exercises, <https://regexone.com>
- [4] RegexLearn, <https://regexlearn.com/learn>
- [5] Interactive Regular Expressions, <https://regex101.com>
- [6] Why Using the Greedy `.*` in Regular Expressions Is Almost Never What You Actually Want, <https://mariusschulz.com/blog/why-using-the-greedy-in-regular-expressions-is-almost-never-what-you-actually->
- [7] Expresiones regulares: cuantificadores perezosos, <https://learntutorials.net/es/cplusplus/topic/1681/expresiones-regulares>