INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

TEORÍA COMPUTACIONAL

PRÁCTICA 03

ALUMNO: HERNÁNDEZ CASTELLANOS CÉSAR URIEL

PROFESOR: ROSAS TRIGUEROS JORGE LUIS

FECHA DE REALIZACIÓN: 30 DE AGOSTO DEL 2017

FECHA DE ENTREGA: 6 DE SEPTIEMBRE DEL 2017

Marco teórico.

Lenguaje regular.

Lenguaje regular. En Lingüística, Matemáticas e Informática y en la jerarquía de Chomsky se refiere a los lenguajes de tipo 3, aquellos que pueden representarse mediante gramáticas regulares, autómatas finitos o expresiones regulares.

Son los lenguajes formales más simples, con los mecanismos de representación y reconocimiento más estudiados. Su aplicación práctica en la teoría y construcción de intérpretes y compiladores de lenguajes de programación o de especificación o formato de información, especialmente como microcomponentes del analizador lexicográfico que detecta los tókenes como constantes numéricas, cadenas de texto, operadores, palabras reservadas (keywords), separadores, etc. Pero también se puede apreciar su uso en máquinas expendedoras, teléfonos públicos, calculadoras y otros artefactos electromecánicos.

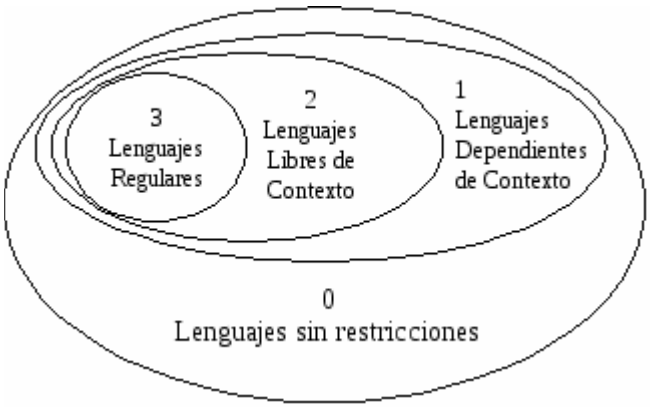

Imagen 1.0 Clasificación de diferentes lenguajes.

Definiciones.

Se dice que un lenguaje es regular si y sólo si se cumple cualquiera de las siguientes proposiciones:

* Tiene al menos una gramática regular G que lo produce.
* Puede ser reconocido por un autómata finito A.
* Existe una expresión regular Er que representa a todas las cadenas de L.
* Dentro de la Jerarquía de Chomsky se refiere a los lenguajes de tipo 3, el subconjunto de lenguajes formales mas restringido dentro de la jerarquía, como se ve en la imagen.

Las expresiones regulares, a menudo llamada también regex, son unas secuencias de caracteres que forma un patrón de búsqueda, las cuales son formalizadas por medio de una sintaxis específica. Los patrones se interpretan como un conjunto de instrucciones, que luego se ejecutan sobre un texto de entrada para producir un subconjunto o una versión modificada

del texto original. Las expresiones regulares pueden incluir patrones de coincidencia literal, de repetición, de composición, de ramificación, y otras sofisticadas reglas de reconocimiento de texto . Las expresiones regulares deberían formar parte del arsenal de cualquier buen programador ya que un gran número de problemas de procesamiento de texto pueden ser fácilmente resueltos con ellas.

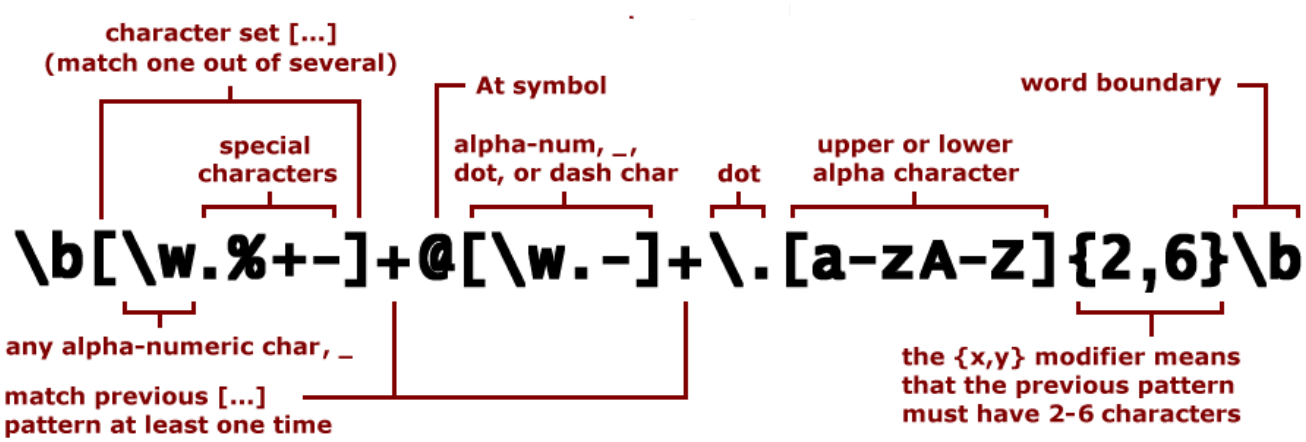

Imagen 1.1 Expresiones regulares.

Componentes de las Expresiones Regulares

Las expresiones regulares son un mini lenguaje en sí mismo, por lo que para poder utilizarlas eficientemente primero debemos entender los componentes de su sintaxis; ellos son:

Literales: Cualquier caracter se encuentra a sí mismo, a menos que se trate de un metacaracter con significado especial. Una serie de caracteres encuentra esa misma serie en el texto de entrada, por lo tanto la plantilla "raul" encontrará todas las apariciones de "raul" en el texto que procesamos.

Secuencias de escape: La sintaxis de las expresiones regulares nos permite utilizar las secuencias de escape que ya conocemos de otros lenguajes de programación para esos casos especiales como ser finales de línea, tabs, barras diagonales, etc.

Clases de caracteres: Se pueden especificar clases de caracteres encerrando una lista de caracteres entre corchetes [], la que que encontrará uno cualquiera de los caracteres de la lista. Si el primer símbolo después del "[" es "^", la clase encuentra cualquier caracter que no está en la lista.

Metacaracteres: Los metacaracteres son caracteres especiales que son la esencia de las expresiones regulares. Como son sumamente importantes para entender la sintaxis de las expresiones regulares y existen diferentes tipos, voy a dedicar una sección a explicarlos un poco más en detalle.

Material y equipo.

El material utilizado en la práctica es el siguiente:

Herramientas de software:

* Oracle VM VirtualBox
* Ubuntu 17.04
* Python 3.6.2
* eric6 Web Browser (QtWebKit)

Herramientas de hardware:

* Computadora personal

Desarrollo de la práctica.

La práctica numero tres consistió en desarrollar un script en python el cual, me generará de manera aleatoria direcciones ip, a la vez que se generó un numero aleatorio asignado a cada dirección ip.

Una vez que la salida del script ha sido guardada en un fichero, se procedió a generar la siguiente expresión regular, con la finalidad de separar las direcciones ip de los números aleatorios, la expresión es la siguiente:

```
grep "[0-9]\
{1,3\}\.[0-9]\
{1,3\}\.[0-9]\
{1,3\}\.[0-9]\
{1,3\}"
numeros.txt
```

Imagen 1.2 Expresión regular usada para separar las direcciones ip.

```
uriel@Uriel-PC:~/Escritorio$ grep "[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}\.[0-9]\{1,3\}" numeros.txt
067|170.096.061.252
202|147.101.251.166
188|123.252.097.138
040|251.018.122.115
178|044.250.195.005
020|142.010.073.241
251|138.043.067.227
224|219.017.168.057
176|057.001.139.244
119|248.094.007.093
uriel@Uriel-PC:~/Escritorio$
```

Imagen 1.3 Salida.

```python
import random



def generarNumeroAleatorio():
    return random.randint(0,255)


def generarTupla():

    for i in range(0,4):
        miNumero = generarNumeroAleatorio()
        cadena = str(miNumero)
        if miNumero<10:
            cadena = "00"+str(miNumero)
        if miNumero>=10 and miNumero <100:
            cadena = "0"+str(miNumero)
    return cadena

def generarDireccionIp():
    cadena = ""
    for i in range(0,4):
        cadena = cadena +"."+ generarTupla()



    return cadena[1:]

def probadorDeAplicacion():
    for i in range(0,10):
        print str(generarTupla())+"|"+generarDireccionIp()

probadorDeAplicacion()
```

Imagen 1.4 Programa en python para generar ip aleatoria.

Conclusiones y recomendaciones.

Las expresiones regulares son patrones utilizados para encontrar una determinada combinación de caracteres dentro de una cadena de texto, también pude observar que no existe una única solución, ya que las expresiones pueden ser tan variadas.

También es posible concluir, que con las expresiones regulares es posible:

* Comprobar si existe un modelo dentro de una cadena.

* Reemplazar texto.

* Extraer una subcadena de una cadena en función de la coincidencia del modelo.

Referencias.

[1] Ccg.unam.mx. (2017). *Expresiones regulares*. [online] Available at: http://www.ccg.unam.mx/~contrera/bioinfoPerl/node18.html [Accessed 6 Sep. 2017].

[2] Ecured.cu. (2017). *Lenguaje regular - EcuRed*. [online] Available at: https://www.ecured.cu/Lenguaje_regular [Accessed 6 Sep. 2017].

[3] Msdn.microsoft.com. (2017). *Usos de las expresiones regulares*. [online] Available at: https://msdn.microsoft.com/es-es/library/101eysae(v=vs.100).aspx [Accessed 6 Sep. 2017].

[4] Omegat.sourceforge.net. (2017). *Capítulo 16. Expresiones regulares*. [online] Available at: http://omegat.sourceforge.net/manual-latest/es/chapter.regexp.html [Accessed 6 Sep. 2017].