TECNOLÓGICO
NACIONAL DE MÉXICO

Proyecto 1. Lenguajes y Autómatas

Proyecto 2

Nombre:

César Alejandro Medina Arredondo

Maestro:

Juan Pablo Rosas Baldazo

¿Qué se hizo?

Se verá tres programas, todos tienen similitudes, ya que trabajan con cadenas de caracteres. Dos de los programas son para reconocer si una dirección de correo electrónico dada está estructurada de forma correcta.

Entre estos dos uno de los programas será enteramente propio, mientras que el otro implementará el uso de una expresión regular para verificar correos. Incluye una prueba para verificar cual es más eficaz. El último programa será una implementación del Algoritmo KMP(Knuth-Morris-Pratt) y probar que funciona.

¿En dónde se hizo?

Se realizó en Python y ejecutado en el mismo Así como la utilización de máquinas las cuales mostrare a continuación; validar formato de un correo electrónico

La maquina que fue utilizada fue:

Lenovo

AMD A8-6410 Radeon Graphics R5 2.00GHZ

Memoria RAM 6.00 GB

Windows 10 Home

¿Cómo se realizó?

Se mostrara paso a paso de acuerdo a cada programa ya que estos tienen una similitud entre si;

Primer programa:

Descripción

Primero se ingresa el correo que se va a verificar mediante un input que guardaremos en

una variable, para después mandarla a la función que lo verificara

```
correo = input("Inserta la direccion de correo electronico")
verificarCorreo(correo)
```

La verificación en si es muy simple, primero verificamos que todos los caracteres son en

letras minúsculas

```
if correo.islower():
```

de lo contrario es una dirección de correo invalido

```
else:
    print("correo falso")
```

a continuación, verificamos que exista una arroba (@)

```
if "@" in correo:
```

si no existe el correo es invalido

```
else:
```

print("correo falso")

si existe separamos la cadena con el @ como marca y las guardamos en variables

usuario,dominio = correo.split("@")

a continuación, revisamos que en la cadena que contiene el dominio exista un punto

if "." in dominio:

si existe entonces el correo está estructurado de forma correcta, de lo contrario es un correo

no valido

if "." in dominio:

print ("Correo valido")

else:

print ("Correo falso")

Segundo programa

Descripción

Primero se ingresa el correo que se va a verificar mediante un input que guardaremos en

una variable, para después mandarla a la función que lo verificara

correo = input("Inserta la direccion de correo electronico")

verificarCorreo(correo)

en validar correo tenemos la expresión regular que nos permitirá revisar si la estructura del

correo es adecuada

'^[(a-z0-9_\-\.)]+@[(a-z0-9_\-\.)]+\.[(a-z)]{2,15}$'

Verificamos si la cadena enviada cuenta con estos requisitos colocándola en un if, de ser

asi entonces es una dirección de correo valida

```
if re.match('^[(a-z0-9\_\-\.)]+@[(a-z0-9\_\-\.)]+\.[(a-z)]{2,15}$',correo.lower()):
    print ("Correo correcto")
```

de lo contrario es una dirección de correo electrónico no valida

```
else:
    print ("Correo incorrecto")
```

Tercer programa

Descripción

El algoritmo KMP consiste en crear una tabla de fallos, para saber cuántos espacios saltar

al haber una discrepancia en la cadena de búsqueda de acuerdo con el patrón que estamos

buscando.

Primero le damos la cadena y el patrón a buscar y lo mandamos

```
 T = input("Inserta el texto.")
    P = input("Inserta el patron a buscar dentro del texto.")
 kmp(P, T)
```

Veamos la construcción de la tabla de fallos, que analiza el patrón para saber cuantos

espacios pueden ser saltados.

```
def tabla_fallos(P):
```

definimos el largo del patrón y la dimensión de la tabla

```
l_p = len(P)
    k = 0
    table = [0] * l_p
```

iniciamos un ciclo que recorra el patrón

```
 for q in range(1, l_p):
```

Se compara un fragmento de la cadena donde se busca con un fragmento de la cadena

que se busca, y esto nos da un sitio potencial para que haya una nueva coincidencia.

```
 while P[k] != P[q] and k > 0:
            k = table[k - 1]
       if P[k] == P[q]:
```

```
            k += 1
        table[q] = k
```

Regresa la tabla con los saltos (1) menos el ultimo carácter

```
 return table[:-1]
```

A continuación, veremos el módulo de búsqueda que usa la tabla de fallos

```
 def kmp(P, T):
```

primero definimos las variables que nos ayudaran

m = 0 salta dentro del texto y encuentra la posicion que se busca

i = 0 indice que recorre la palabra

pos = 0 es para saber cuando la palabra no existe

l_p = len(P)largo del patron

l_t = len(T)largo del texto

después se confirma que el texto es mas grande que el patrón que se buscara

```
 if(l_t >= l_p):
```

se genera la tabla de fallos

```
 tabla = tabla_fallos(P)
```

empieza ciclo para recorrer las posiciones del texto siempre que la posision de salto y el

índice no exedan el largo del texto y patron respectivamente

```
 while((m<l_t) and (i<=l_p)):
```

busca las posision en que hay una coincidencia e incrementa el índice

```
if(P[i] == T[m+i]):
            if(i == (l_p-1)):
                pos = pos + 1
                print ("esta en la posicion %s" %m)
                return
            i= i+1
```

de lo contrario el índice salta a la posision indicada en la tabla de fallos

```
 else:
        m = m + i - tabla[i]
        if(i>0):
           i = tabla[i]
```

si la posision es igual a 0, no se encontró coincidencia

```
 if pos == 0:
            print ("no se encuentra")
```

¿Para qué se realizó?

Para observar el diferente comportamiento que se tiene al correr los dos programas y la relación que tiene entre ambos. Asi como el tercer programa es un poco diferente a los otros 2. Al hacer todo esto vimos como las expresiones regulares son mucha ayuda en los procesos de los cuales necesitan optimizar recursos.

PRUEBAS PROGRAMA 1

Para probar el tiempo de ejecución de la aplicación, usare diferente longitud en las cadenas.

Pruebas de longitud de la cadena

Las pruebas de longitud.

Cadena	Longitud	Resultado
rubius_ex@hotmail.com	17	1.567763090
sinh@hotmail.com	25	1.90247249
Alex1325@hotmail.com	31	1.669835

Como podemos observar, la longitud de la cadena no es relevante, ya que la variación en segundos es muy baja.

PRUEBAS PROGRAMA 2

Para probar el tiempo de ejecución de la aplicación, usare diferente longitud en las cadenas.

Pruebas de longitud de cadena Primero probaremos cómo reacciona el sistema cuando se le ingresan cada vez más datos.

Cadena	Longitud	Resultado
rubius_ex@hotmail.com	17	1.53010129
sinh@hotmail.com	25	1.4993832
Alex1325@hotmail.com	31	1.667841

Al incrementar la longitud de la cadena, se nota un ligero incremento en el tiempo de ejecución.

PRUEBAS PROGRAMA 3

Para probar el tiempo de ejecución de la aplicación, usare diferente longitud en las cadenas.

Pruebas de longitud de cadena

Primero probaremos cómo reacciona el sistema cuando se incrementa el tamaño de la cadena.

Cadena	Patron	Longitud	Resultado
aaaaabbaab	baab	10	6.11112542
abababababaabc	aabc	14	6.12733530
bbbabbababa	baba	17	7.106204195
10101110101101110	1110	21	7.119913883
casascasacvasc	vasc	25	7.13163971