INSTITUTO TECNOLOGICO DE NUEVO LEON

Carrera:

Ing. Sistemas Computacionales

MATERIA:

Lenguajes autómatas 1

Trabajo:

Reporte de Proyecto 2

Profesor:

Ing. Juan Pablo Rosas Baldazo

Alumno:

Eduardo Martínez Martínez

15690231

¿Qué se hizo?

Este documento se verán 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 MEDIANTE MÉTODO PROPIO

El equipo en que se harán las pruebas es un laptop HP Pavilon x360

- Procesador i5
- Graficos Rendon ddr 5 Graphics
- Windows 10 Home de 32 bits
- 6 GB de memoria RAM

VALIDAR FORMATO DE CORREO ELECTRÓNICO MEDIANTE UNA EXPRESIÓN REGULAR.

El equipo en que se harán las pruebas es un laptop HP Pavilon x360

- Procesador i5
- Graficos Rendon ddr 5 Graphics
- Windows 10 Home de 32 bits
- 6 GB de memoria RAM

ALGORITMO KMP

El equipo en que se harán las pruebas es un laptop HP Pavilon x360

- Procesador AMD A8-6410 APU 2.00GHz
- Graficos Rendon R5 Graphics
- Windows 10 Home de 32 bits
- 4 GB de memoria RAM

¿Cómo se realizó?

se les mostrara paso a paso de acuerdo a cada programa ya que tiene similitudes 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.

Así como también el tercer programa es un poco diferente de los otros 2 programas.

Gracias a la realización de todo esto, nos damos cuenta que las expresiones regulares son mucha ayuda en procesos que se necesitan optimizar recursos, en cuanto al tiempo de ejecución de un programa y el costo de trabajo en algún proceso industrial.

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
ultim@hotmail.com	17	1.567763090133667
Ultimátum_10f@hotmail.com	25	1.9024724960327148
Ultimátum_10forever@hotmail.com	31	1.669835090637207

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
ultim@hotmail.com	17	1.5301012992858887
Ultimátum_10f@hotmail.com	25	1.4993832111358643
ultimatum_10forever@hotmail.com	31	1.6678433418273926

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

PRUEBAS PROGRAMA 2

Pruebas de cantidad de iteraciones

Esta vez tomaremos uno de los casos anteriores y lo repetiremos 10 veces para observar las diferencias entre el tiempo de ejecución con los mismos parámetros.

Cadena	longitud	Iteración	Resultado
ultimatum@hotmail.com	21	1	1.4516689777374268
ultimatum@hotmail.com	21	2	1.0698623657226562
ultimatum@hotmail.com	21	3	1.3995728492736816
ultimatum@hotmail.com	21	4	0.9732472896575928
ultimatum@hotmail.com	21	5	1.052912712097168
ultimatum@hotmail.com	21	6	1.180114507675171
ultimatum@hotmail.com	21	7	1.1998951435089111
ultimatum@hotmail.com	21	8	1.0866975784301758
ultimatum@hotmail.com	21	9	0.995703935623169
ultimatum@hotmail.com	21	10	1.249755859375

Al igual que en la prueba de iteraciones de la otra función, podemos observar que, aun cuando los parámetros son iguales, el procesador los ejecutara a diferente velocidad dependiendo de que este haciendo en ese momento.

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	Patrón	longitud	Resultado
aaaaaaaaab	aaab	10	6.111125421524048
abcabcabcabacf	bacf	14	6.1273353099823
bbbbbbaabbbaaab	aaab	17	7.106204195022583
1001010111011000123	0123	21	7.1199138832092285
ccnmcceerteehedddddaaab	aaab	25	7.131639719009399