Lenguajes y Autómatas

Juan Pablo Rosas Baldazo

Reporte de proyecto 1

Alfonso Guerrero Contreras
de control16480211

10/09/2018

Introducción

Un alfabeto es un conjunto de caracteres con los cuales se pueden formar palabras, las cuales llamamos lenguaje.

Es posible crear una cantidad enorme de combinaciones (palabras) para cada alfabeto. Así cómo es posible también deducir un alfabeto a partir de un conjunto de palabras.

En este documento se detallará el desarrollo de un programa, codificado en Python, que permite leer un alfabeto y generar una lista de palabras (lenguaje) de tamaño variable.

Su segunda función es, determinar un alfabeto a partir de un lenguaje, leyendo el alfabeto ingresado y formando palabras.

Ambas funciones mostraran sus resultados impresos en pantalla.

Descripción

Generar un lenguaje a partir de un alfabeto

Se me sugirió empezar creando un generador de cadenas aleatorias, para esto cree una lista con las vocales y puse barios ciclos para que me diera todas las combinaciones posibles. Esto, aunque funciono, no era optimo, por lo que investigando encontré la función "*choice*" de la librería "*random*" (que ya se me había recomendado importar por parte de mi profesor).

Leyendo en la página de Python sobre como se usa, me di cuenta de que podía mandarle una cadena o lista y tomaría uno de los caracteres, el cual yo concatenaría en una cadena dentro de una variable.

```
GeneradorLenguaje (n,alfabeto)
    lenguaje = []
    palabra=""
   palabra=palabra.join(random.choice(alfabeto) for _ in range(longitud))
```

El siguiente paso consistió en agregar la *palabra* aleatoria a una lista que contendrá el lenguaje, sin embargo, para que las palabras añadidas no se repitieran implemente la condición, "si la palabra generada no está en la lista, entonces se añade a esta y se borra el contenido de la variable que tenía la palabra aleatoria, de lo contrario, solo se borra la palabra generada".

```
if palabra not in lenguaje
        lenguaje.append(palabra)
        control = control +1
        palabra = ""
    else
        palabra = ""
```

Ya tenia un generador de cadenas al azar, ahora necesitaba que se repitiera el proceso tantas veces como fuera necesario. Para lograrlo use una condición de ciclo "while", y un contador para contar las iteraciones, de forma que sean las mismas que se pidieron.

```
while control != n
    longitud=random.randint(1, 6)
    palabra=palabra.join(random.choice(alfabeto) for _ in range(longitud))
    if palabra not in lenguaje:
```

```
            lenguaje.append(palabra)
            control = control +1
            palabra = ""
        else:
            palabra = ""
```

Para terminar esta función solo faltaba hacer que la longitud de palabras generadas fuera aleatorio, para lograrlo, basto con declarar una variable que usaría para determinar la longitud de la palabra e inicializarla con la función “random.randint(n,m)”, que encontré mientras leía la biblioteca “random”. Esta devuelve un numero al azar de entre un rango dado (1,6).

```
longitud=random.randint(1, 6)
```

Con esto tenía una función que generaba tantas palabras como quisiera, de tamaño aleatorio, que son guardadas en una lista a partir de un alfabeto. Por lo que para terminar esta primera función solo faltaba, permitir la entrada de datos necesarios (el alfabeto y la cantidad de palabras).

Cree una nueva función que serviría para ejecutar ambas funciones requeridas para este proyecto, así como la captura de los datos necesarios, los cuales son enviados hacia sus respectivas funciones.

```
ingresarDatos()
    prueba = input(" Ingresa (a) si deseas ontener un alfabeto a partir de un lenguaje
    \n Ingresa (l) si deseas obtener un lenguaje a partir de un alfabeto")
    if prueba == "l":
        alfabeto = input("Ingresa las letras del alfabeto a utilizar.")
        n = int(input("Ingresa la cantidad de palabras que generara el lenguaje."))
        print(GeneradorLenguaje(n,alfabeto))
```

Generar un alfabeto a partir de un lenguaje

Para esta funcionalidad, inicie creando la función “GeneradorAlfabeto”, que recibe una lista con las palabras que conforman el lenguaje.

```
GenerardorAlfabeto(lenguaje):
    alfabeto = []
```

El siguiente paso consistió en juntar todas las palabras en una sola cadena para facilitar su procesamiento, usando la función {“”.join(lista)}, que encontré al hacer un programa previo, esta junta todos los elementos en una lista que están separados por comas.

```
frase = "".join(lenguaje)
```

Después en un ciclo “for” recorre la frase y dentro esta la condición, “Si el carácter actual no se encuentra el alfabeto (lista creada prebiamente), esta se añade”, para terminar, se regresa el alfabeto.

```
for i in frase:
    if i not in alfabeto:
        alfabeto.append(i)
```

Esto daba el resultado deseado, sin embargo, el resultado estaba en desorden. De modo que implemente la función “sort” al alfabeto para que se acomodaran en orden ascendiente.

```
alfabeto.sort()
```

Por último, cree un menú para que se decidiera que funcionalidad usar, con un if para cada una, si ingresan a, te dará el alfabeto a partir del lenguaje. Si ingresan l, te dará un lenguaje a partir de un alfabeto.

```
if prueba == "a":
    lenguaje = input("Ingresa las palabras que deseas utilizar separadas por
    esacios.")
    lenguaje = lenguaje.split()
    print(GenerardorAlfabeto(lenguaje))
```

Pruebas

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

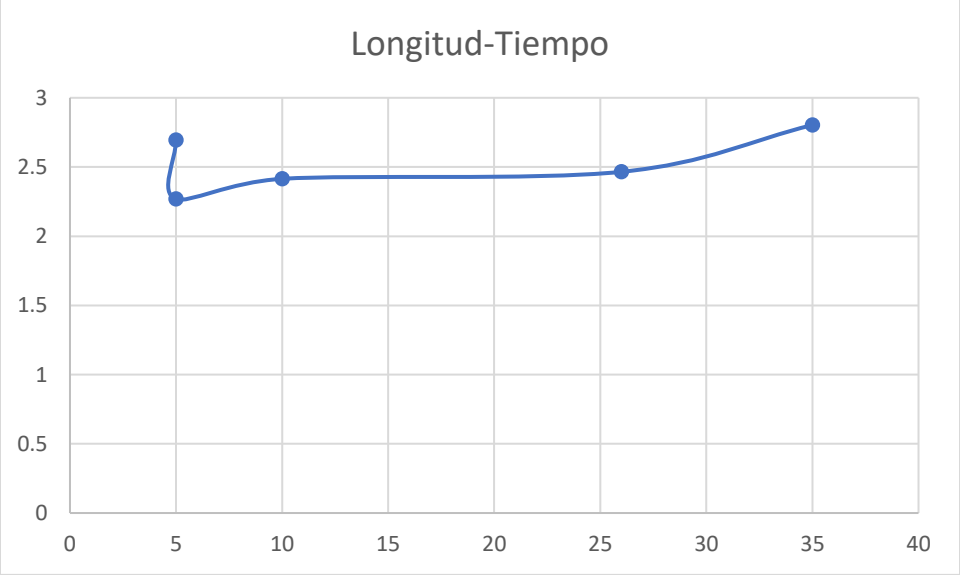
- Procesador AMD A8-6410 APU 2.00GHz
- Graficos Rendon R5 Graphics
- Windows 10 Home de 64 bits
- 4 GB de memoria RAM

Para probar el tiempo de ejecución de la aplicación, usare varias secuencias de caracteres y palabras de diferente longitud y repeticiones:

Pruebas de longitud de cadena para la aplicación de generar un alfabeto

Las pruebas de longitud.

Cadena	longitud	Resultado
a e i o u	5	2.6942105
1 2 3 4 5	5	2.26825
12345 abcde	10	2.4146552
12cd5 ab34e aed 531 12345 abcde	26	2.4650972
123456789 abcdefg hijklmn opqrst uvwxyz	35	2.8030076

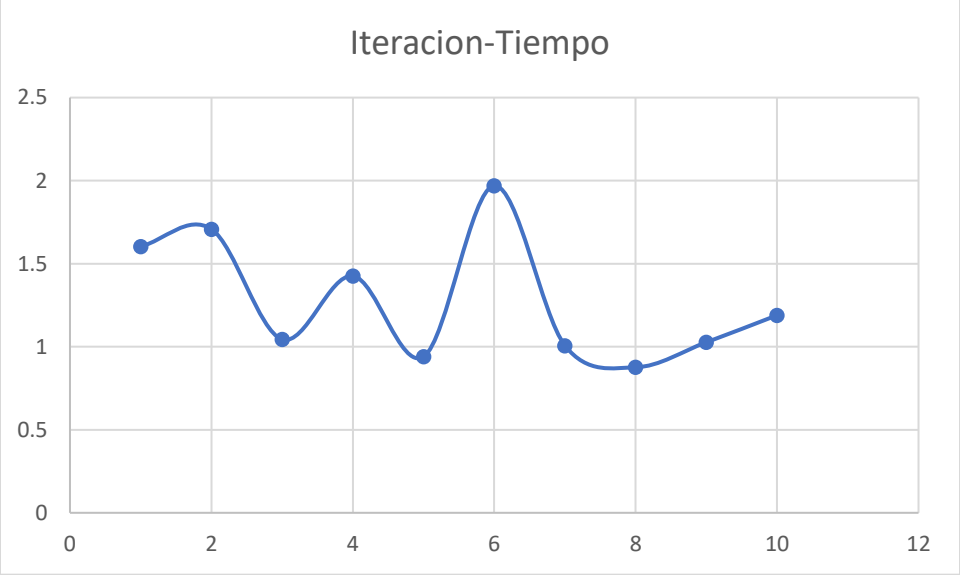


Como podemos observar, la longitud de la cadena no es relevante, ya que la variación en segundos es muy baja.

Pruebas de cantidad de iteraciones para la aplicación de generar un alfabeto

Esta vez tomaremos uno de los casos anteriores y lo repetiremos 20 veces para observar las diferencias entre el tiempo de ejecución con los mismos parámetros.

Cadena	longitud	Iteración	Resultado
12345 abcde	10	1	1.6035737991333008
12345 abcde	10	2	1.7078142166137695
12345 abcde	10	3	1.0443944931030273
12345 abcde	10	4	1.4260883331298828
12345 abcde	10	5	0.9399464130401611
12345 abcde	10	6	1.9689679145812988
12345 abcde	10	7	1.0069208145141602
12345 abcde	10	8	0.8761608600616455
12345 abcde	10	9	1.026745080947876
12345 abcde	10	10	1.1894574165344238



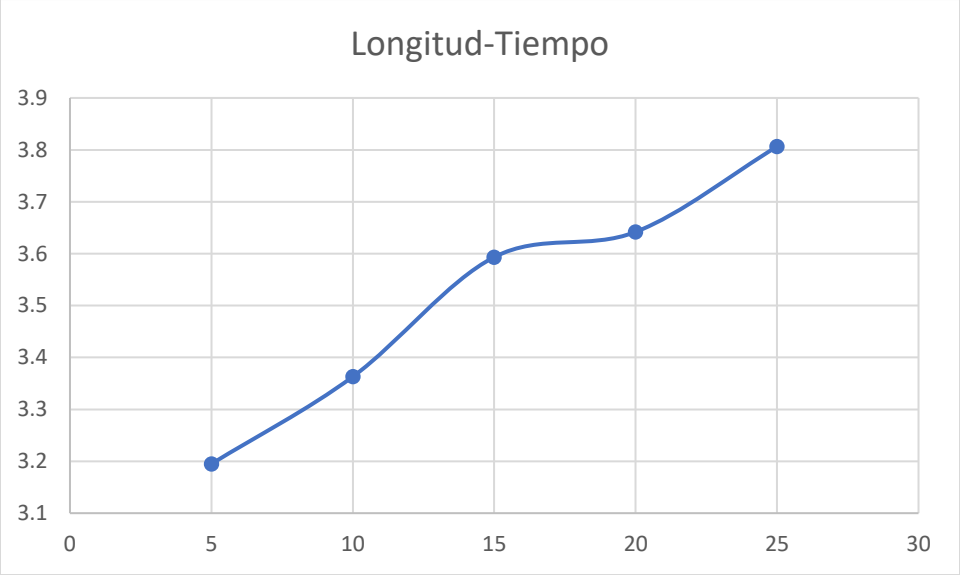
- En esta prueba podemos observar que, aun cuando los parámetros son iguales, el procesador los ejecutara a diferente velocidad dependiendo de que este haciendo en ese momento.

Pruebas de longitud de cadena para la aplicación generar un lenguaje

Primero probaremos como reacciona el sistema cuando se le ingresan cada vez más datos.

Cadena	# palabras	longitud	Resultado
12345	20	5	3.1948299407958984
abcdefghij	20	10	3.3629891872406006
0123456789abcde	20	15	3.593363046646118
abcdefghij!”#$%&/()=	20	20	3.6419031620025635
0123456789abcdefghijklmno	20	25	3.806324005126953

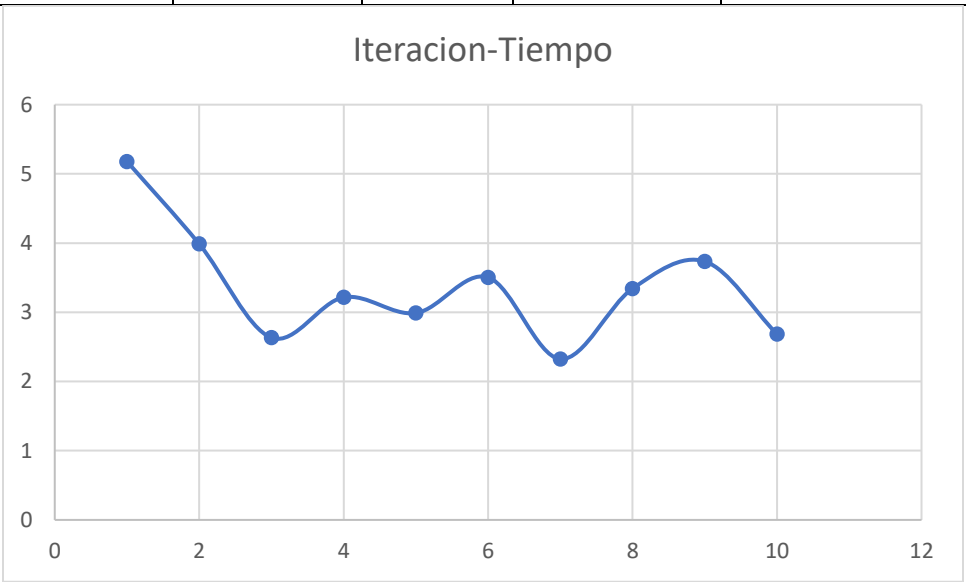
- Al incrementar la longitud de la cadena (alfabeto), se nota un ligero incremento en el tiempo de ejecución.



Pruebas de cantidad de iteraciones para la aplicación de generar un lenguaje

Esta vez tomaremos uno de los cazos anteriores y lo repetiremos 20 veces para observar las diferencias entre el tiempo de ejecución con los mismos parámetros.

Cadena	# palabras	longitud	Iteración	Resultado
0123456789abcde	20	15	1	5.1792988777160645
0123456789abcde	20	15	2	3.987222909927368
0123456789abcde	20	15	3	2.6328537464141846
0123456789abcde	20	15	4	3.216392993927002
0123456789abcde	20	15	5	2.9891207218170166
0123456789abcde	20	15	6	3.5046958923339844
0123456789abcde	20	15	7	2.3242926597595215
0123456789abcde	20	15	8	3.342362403869629
0123456789abcde	20	15	9	3.734771251678467
0123456789abcde	20	15	10	2.6841862201690674



- 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.

Conclusión

El poner dos funcionalidades en un solo programa, lo volvió complejo, al tener que diferenciar entre que se desea hacer.

El uso de las funciones “.join(lista)” y “.split()”, fueron resultado de la investigación necesaria para realizar el programa que determinaba si una frase es un palíndromo, implementarlas en este proyecto me ayudo a entender mejor su funcionamiento con referencias basadas en experiencia real.

En general la realización de este proyecto es una excelente practica para desarrollar las habilidades de investigación, comprensión e implementación de métodos de un lenguaje de programación dentro de un proyecto.

Fuentes

https://docs.python.org/3/library/random.html

https://docs.python.org/3/library/

https://docs.python.org/3/library/random.html