Universidad de Lima
Facultad de Ingeniería de Sistemas
Estructura de Datos y Algoritmos

ANÁLISIS DE ALGORITMOS DE BÚSQUEDA DE PATRONES

Eduardo Enrique Ramón Zuta
Código 20201726

Profesor
Hernán Nina Hanco

Sección: 501

Lima – Perú
Noviembre de 2021

TABLA DE CONTENIDO

ÍNDICE DE FIGURAS

1. Resumen:

En el presente trabajo, se mostrará un análisis de dos algoritmos pertenecientes a la familia de búsqueda de patrones, entre ellos el “Z Algorithm” y “Anagram Pattern Search”. De esta manera, se señalará sus aplicaciones, su implementación en el lenguaje de programación “Python” y su efectividad frente a distintas tareas.

2. Introducción

Tareas como un corrector ortográfico o un filtro de spam suelen ser herramientas cotidianas que usamos diariamente, por ejemplo, el gran servicio de correo Gmail otorga un filtro de spam a sus usuarios, y el programa Word de Microsoft incluye en sus versiones más actuales un corrector ortográfico. De esta manera, estas herramientas y muchas más, pueden ser resueltas mediante el uso de algoritmos de búsqueda de patrones. Por ello, en las siguientes secciones se mostrará el estado del arte de los algoritmos, los problemas que solventa, una metodología para analizarlos cuantitativamente, los resultados de esta y finalmente las conclusiones del trabajo.

3. Estado del Arte:

Para comprender a los algoritmos planteados, se utilizan los papers, los cuales brindan la información necesaria para entender dichos algoritmos, su planteamiento y sus estructuras, en el caso de "Z Algorithm", este algoritmo se centra en encontrar apariciones de un mismo patrón en un texto de tiempo lineal, donde la longitud del texto sería (n), en ese caso sería O(m+n) con complejidad de espacio lineal, otro algoritmo planteado es el "Anagram Pattern Search", es español: "Búsqueda de patrones de anagramas", este algoritmo es el proceso de comprobar una secuencia percibida en una cadena de caracteres, esto significa que para los patrones de coincidencia de cadenas, las cadenas se buscan dentro de cadenas más largas o textos. Si hay una cadena de patrón "p" y la cadena de texto "S", el problema es encontrar un conjunto de patrones "p" en "S" o no. Si aparece "p", entonces su posición debe indicarse en "S". Estos referentes, papers, facilitan el comprender los algoritmos y sus estructuras, ya que toda esta información ya ha sido compilada y sistematizada para que el usuario al acceder a estos pueda entender con mayor facilidad la información.

3.1. El problema de investigación:

El problema de esta investigación se genera la repetición de una secuencia de letras o palabras, estos algoritmos permiten encontrar la ubicación y la cantidad de veces que esta ha sido repetida, es decir estos algoritmos se hacen cargo de encontrar patrones de texto, dentro de textos más grandes. En uno de los papers, se indica que los "Z Algorithm" tienen aplicaciones, una es en la que se pueden encontrar ocurrencias de patrones "p" dentro de textos "T", con la forma O(length(T) + lenght(p)). Un ejemplo del Z Algorithm:

> If P = aba and T = bbabaxababay then P occurs in T starting at locations 3, 7, and 9. Note that two occurrences of P may overlap, as illustrated by the occurrences of P at locations 7 and 9. (Gusfield, 1997)

3.2. Algoritmos que resuelven el problema:

Un ejemplo donde se apliquen estos algoritmos, es el caso de un informe titulado "Algoritmos de búsqueda de subcadenas para encontrar semejanzas en cadenas numéricas" , donde se implementaron distintos algoritmos de búsqueda de sub-cadenas en cadenas numéricas dadas, en este informe se demuestra como los algoritmos han sido aplicados a medidas de variaciones que experimentan los troncos de los árboles, se implementaron dos tipo de algoritmos: algoritmos generales de cadenas, los cuales buscan características que se producen en el tronco

y algoritmo de búsqueda de sub-cadenas, los cuales buscan un patrón en las cadenas que se generan. Y ambos tipos de algoritmos se implementan en una aplicación gráfica, donde el usuario puede elegir que algoritmo usará y las características con las que lo usa. Y algo que señalar en el informe, es que se encuentran diferentes algoritmos que resuelven los mismos problemas, pero no han sido implementados en nuestro informe: "el algoritmo Shift-Add tarda un tiempo razonable para cualquier problema estudiado y que el algoritmo Boyer Moore es razonablemente bueno en el caso general que contemplamos y cuando hay pocas ocurrencias en las medidas del patrón buscado." (Coronado Sanz, 2018)

3.3. Comparaciones de Algoritmos en general:

En el artículo titulado "Analysis and Performance Evaluation of Selected Pattern Matching Algorithms" por Princewill Aigbe y Emmanuel Nwelih, se realiza un análisis cuantitativo de 5 algoritmos para la búsqueda de patrones, entre ellos tenemos a, "Naïve Pattern Matching Algorithm", "Rabin-Karp Pattern Matching Algorithm", "Knuth-Morris-Pratt pattern matching algorithm", "Boyer-Moore pattern matching algorithm" y "Anagram substring pattern matching algorithm". De esta manera, para el análisis de cada algoritmo, se utilizó dos criterios claves, su tiempo de ejecución y el número de comparaciones empleadas para encontrar un patrón dado un texto. Asimismo, los algoritmos fueron ejecutados en un mismo computador. Consecuentemente, los resultados obtenidos fueron mostrados en gráficas de dos y tres dimensiones. (Princewill, 2021)

4. Marco teórico:

Dada una secuencia de letras (o en términos sencillos, una palabra) y un texto, los algoritmos de búsqueda de patrones permiten encontrar la ubicación y la cantidad de ocurrencias de esta palabra en todo un texto. Por simplicidad, se habló de palabras, pero también es aplicable a una secuencia de estas (una oración). De esta manera, hablando en términos más generales, estos algoritmos se encargan de encontrar la ocurrencia de un objeto, dentro de un conjunto de objetos más grande. Concretamente, se hablará de los algoritmos "Z Algorithm" y "Anagram Pattern Search".

En base a la idea dada anteriormente, estos algoritmos son empleados en una variedad de tareas que serán expuestas a continuación. La detección de plagio, ya que se podrá determinar la similaridad entre distintos informes para verificar si el trabajo es original o es un plagio. En trabajos de informática es importante el análisis de secuencias de ADN. Se usa en la criminología para detectar frases de interés en alguna investigación. Además, es utilizado en los correctores ortográficos mediante el uso de un porcentaje similaridad con una palabra para corregir al usuario. También como un detector de spam, ya que es usual las palabras claves que utilizan los mensajes enviados con fines publicitarios. En las bases de datos, dada alguna palabra clave, sirve para organizar la información.

En términos de análisis de algoritmos con la notación Big-O, el "Anagram Pattern Search" tiene una complejidad lineal de O(n), mientras que "Z Algorithm" tiene O (m + n), donde m es el largo de la palabra y n del texto. (GeeksforGeeks, 2020)

5. Metodología:

Debido a que nuestro objetivo es analizar algoritmos de búsqueda de patrones, nuestra metodología trata de dos pasos. El primero de ellos, consistirá de un análisis teórico, en el cual se determinará la complejidad de los algoritmos con la notación Big-O. El segundo y último paso, consta de un análisis experimental, en donde se pondrán a prueba los algoritmos con textos de distintos tamaños (en MB) para conseguir su tiempo de ejecución correspondiente.

Una vez que ya se tenga el algoritmo desarrollado en Python, no hay mucho trabajo detrás del primer paso, ya que solamente se haría el análisis de complejidad Big-O. Por contraparte, en el análisis experimental, una pieza fundamental son los datos con los que se pondrán a prueba los algoritmos en cuestión. Para ello, se recurrió a Kaggle, una plataforma pública donde cualquier usuario puede publicar data sets (Kaggle: Your Machine Learning and Data Science Community, s.f.). Específicamente, se utilizó un data set creado por Kondalarao Vonteru que consiste de 4515 ejemplos que contiene el encabezado de una noticia y un texto resumen sobre aquella (Vonteru, 2019). De esta manera, para alimentar a nuestros algoritmos, los resúmenes de las noticias fueron concatenadas en un solo texto con un tamaño de 33.58 MB. Consecuentemente, se extrajo aleatoriamente, el 25, 50 y 75% de las palabras, resultando 4 unidades de análisis (25%, 50% 75% y 100% que tienen un tamaño de 8.39, 16.79, 25.19 y 33.58 MB respectivamente). Posteriormente, aún falta determinar los patrones (palabras en este caso) con los que se pondrá a prueba los algoritmos, para ello, de cada unidad de análisis, se extrajo 5 palabras aleatorias. De esta forma, en el siguiente apartado se mostrarán los resultados correspondientes.

6. Resultados:

Primeramente, los algoritmos fueron implementados en la plataforma Google Colaboratory, en el cual se maneja el lenguaje de programación Python. Refiriéndonos a las especificaciones del computador, nos proveyeron 107.72 GB de disco, 12.69 GB de RAM y un procesador Intel(R) Xeon(R) de 2.20GHz. De esta manera, las implementaciones de los algoritmos se pueden apreciar en la figura 4 y 5, en referencia al Z Algorithm y Anagram Algorithm respectivamente. Cabe destacar que ambos algoritmos fueron extraídos de la página Geeks for Geeks (Link de donde se encontró la información acerca de "Z Algorithm": https://www.geeksforgeeks.org/z-algorithm-linear-time-pattern-searching-algorithm/) y (Link de donde se encontró la información acerca de "Anagram Algorithm": https://www.geeksforgeeks.org/anagram-substring-search-search-permutations/). Por parte del análisis teórico, "Z Algorithm" y "Anagram Pattern Search" tienen una complejidad de O (m + n) y O(n) respectivamente, donde m es el largo del patrón y n del texto. (GeeksforGeeks, 2020)

Figura 1. Implementación del "Z algorithm"

```
[ ]  def getZarr(string, z):
       n = len(string)
       l, r, k = 0, 0, 0
       for i in range(1, n):
         if i > r:
           l, r = i, i
           while r < n and string[r - l] == string[r]:
             r += 1
           z[i] = r - l
           r -= 1
         else:
           k = i - l
           if z[k] < r - i + 1:
             z[i] = z[k]
           else:
             l = i
             while r < n and string[r - l] == string[r]:
               r += 1
             z[i] = r - l
             r -= 1
```

```
[ ]  def Z(text, pattern):
       concat = pattern + "$" + text
       l = len(concat)

       z = [0] * l
       getZarr(concat, z)

       occurrences = []
       for i in range(l):
         if z[i] == len(pattern):
           occurrences.append(i - len(pattern) - 1)
       return occurrences
```

Fuente: Elaboración propia

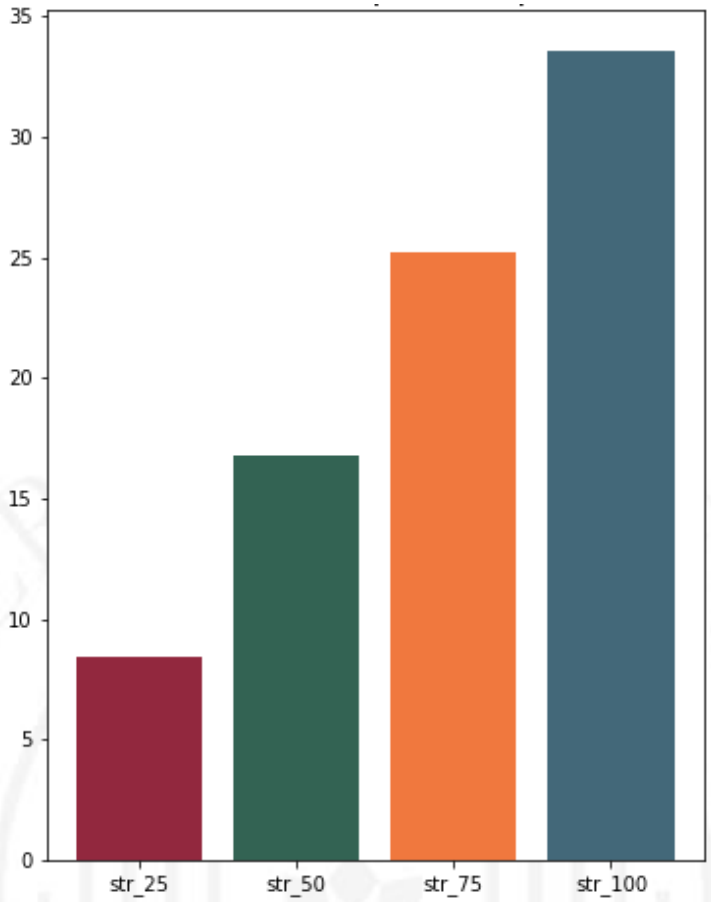
Figura 2. Implementación del "Anagram Search"

```
[ ]  MAX=256

     def compare(arr1, arr2):
       for i in range(MAX):
         if arr1[i] != arr2[i]:
           return False
       return True
```

```
[ ]  MAX=256

     def Anagram(pattern, text):

       M = len(pattern)
       N = len(text)
       occurrences =[]

       countP = [0]*MAX
       countTW = [0]*MAX

       for i in range(M):
         (countP[ord(pattern[i]) ]) += 1
         (countTW[ord(text[i]) ]) += 1

       for i in range(M,N):
         if compare(countP, countTW):
           occurrences.append(i-M)

         (countTW[ ord(text[i]) ]) += 1 #Añadir un caracter
         (countTW[ ord(text[i-M]) ]) -= 1 #Eliminar el primer caracter

       if compare(countP, countTW):
           occurrences.append(N-M)
```

Fuente: Elaboración propia

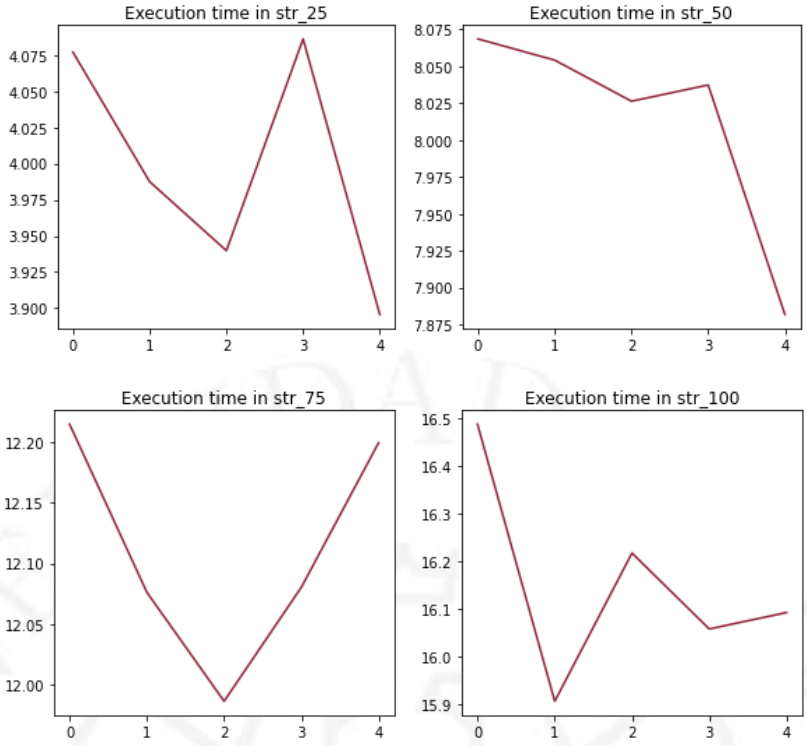
Figura 3. Tamaño en MB de las unidades de análisis.



Fuente: Elaboración propia

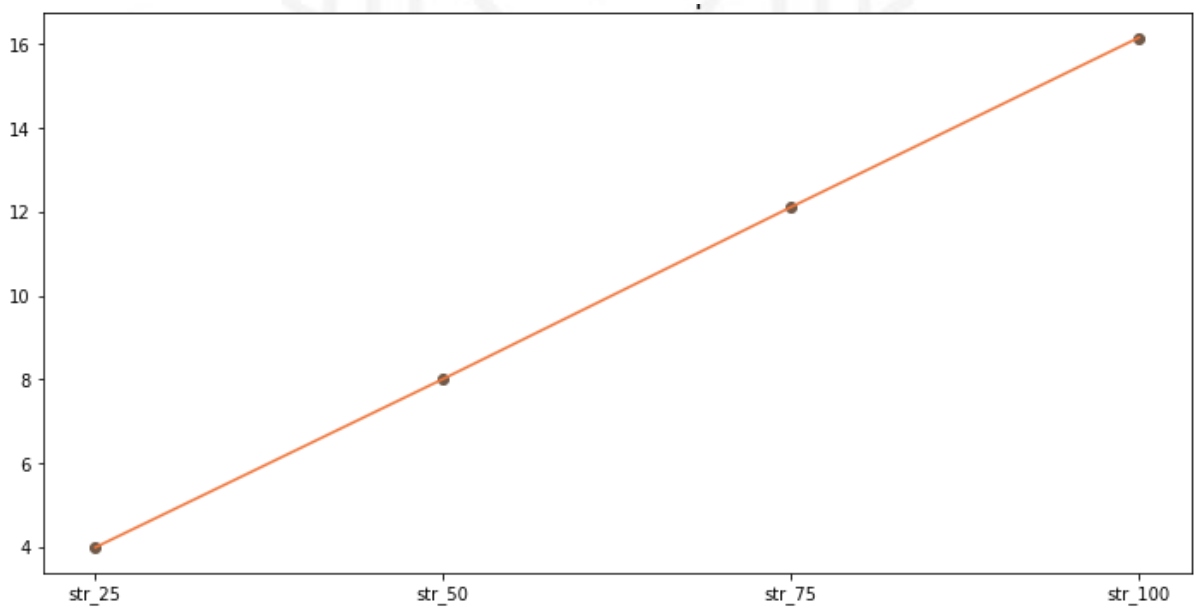
Como fue dicho anteriormente, los datos fueron extraídos de Kaggle y a su vez, divididos en 4 unidades de análisis, referentemente al 25%, 50% 75% y 100% de los datos originales que tienen un tamaño de 8.39, 16.79, 25.19 y 33.58 MB respectivamente, estas se pueden apreciar en la Figura 3.

Figura 4. Tiempo de ejecución del "Z algorithm" con cada unidad de análisis



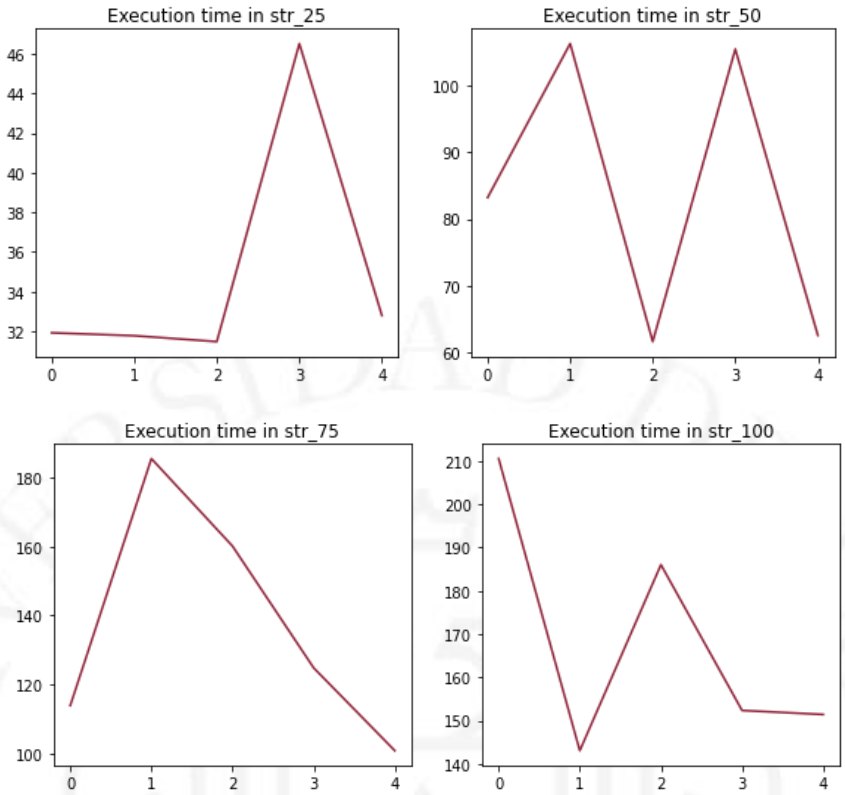
Fuente: Elaboración propia

Figura 5. Promedio simple del tiempo de ejecución con el "Z algorithm" con cada unidad de análisis



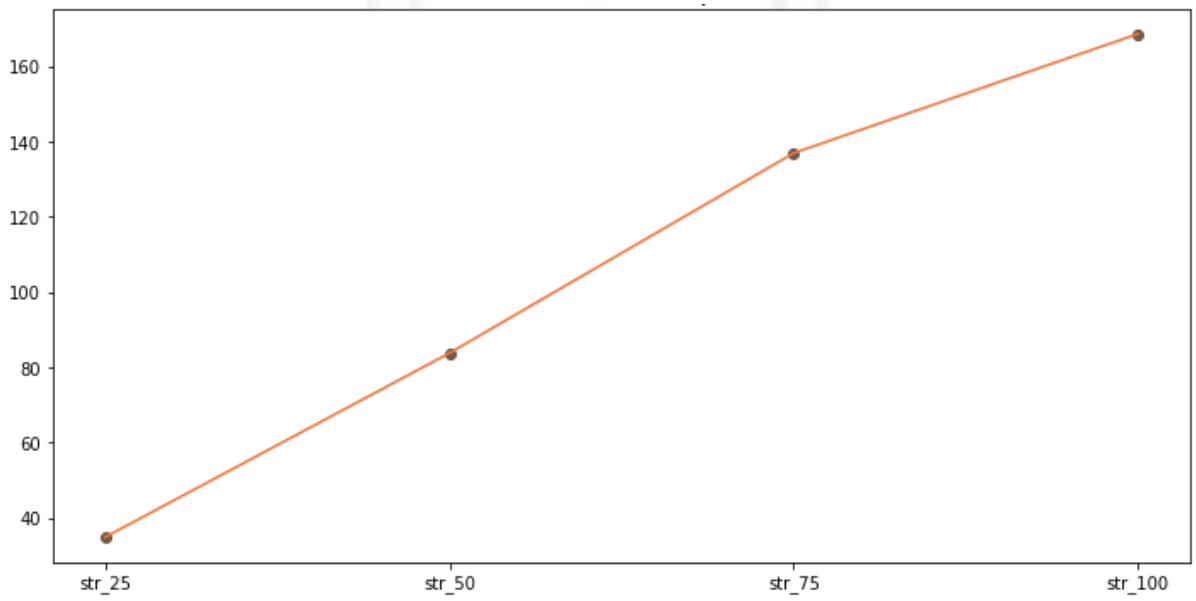
Fuente: Elaboración propia

Figura 6. Tiempo de ejecución del "Anagram Search" con cada unidad de análisis



Fuente: Elaboración propia

Figura 7. Promedio simple del tiempo de ejecución con el "Anagram Search" con cada unidad de análisis

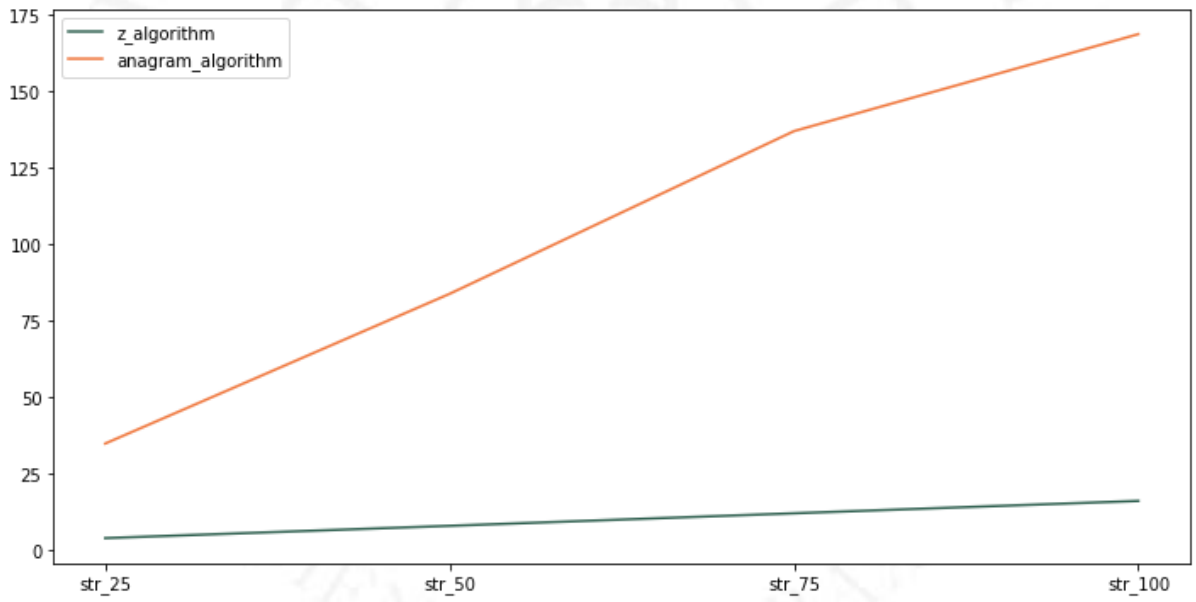


Fuente: Elaboración propia

Refiriéndonos al “Z algorithm”, de cada unidad de análisis se extrajo 5 palabras aleatorias y consecuentemente se obtuvo el tiempo de ejecución por cada una de ellas, estos resultados pueden ser vistos gráficamente en la Figura 4. De esta manera, el promedio simple por cada unidad de análisis está representado en la Figura 5.

Tal cual fue descrito con el “Z algorithm”, se realizó el mismo procedimiento con “Anagram Search”. Los resultados están en la Figura 6 y la Figura 7.

Figura 8. Tiempo de ejecución de “Z algorithm” y “Anagram Search” por cada unidad de análisis.



Fuente: Elaboración propia

El resultado de ambos algoritmos, hablando sobre el tiempo de ejecución, se aprecian en la Figura 8.

7. Conclusiones:

- Se pudo lograr un análisis significativo y verificar la complejidad de cada algoritmo. Como fue descrito en el análisis teórico, ambos algoritmos presentaban una complejidad lineal, y esta se puede apreciar individualmente en la Figura 5 y la Figura 7, y ya que a medida que crece el tamaño de la unidad de análisis (un texto en nuestro caso) el tiempo de ejecución también crece linealmente.
- El “Z algorithm” se desempeña mucho mejor que el “Anagram Search”, esto se puede ver en la Figura 8, de esta manera, hablando en términos cuantitativos, el tiempo promedio de ejecución del “Z algorithm” cuando se utilizó la unidad de análisis 100% (33.58 mb) duró 16.15 segundos, mientras que el “Anagram Search” 168.68 segundos.

Bibliografía

Bourque, P., & Fairley, R. (2014). *Guide to the Software Engineering Body of Knowledge.* Obtenido de Swebok: https://cs.fit.edu/~kgallagher/Schtick/Serious/SWEBOKv3.pdf

Coronado Sanz, B. (2018). *Algoritmos de búsqueda de subcadenas para encontrar semejanzas en cadenas numéricas.* Madrid, España: Universidad Complutense de Madrid.

Foundation, O. (18 de Julio de 2018). *Z Algorithm function*. Obtenido de OpenGenus IQ: Computing Expertise & Legacy: https://iq.opengenus.org/z-algorithm-function/

GeeksforGeeks. (03 de Setiembre de 2020). *Applications of String Matching Algorithms*. Obtenido de Geeks for Geeks: https://www.geeksforgeeks.org/applications-of-string-matching-algorithms/

Gupta, P. (07 de Julio de 2021). *Anagram Substring Search (Or Search for all permutations)*. Obtenido de Geeks For Geeks: https://www.geeksforgeeks.org/anagram-substring-search-search-permutations/

Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences COMPUTER SCIENCE AND COMPUTATIONAL BIOLOGY.* New York, USA: Cambridge University Press.

Kaggle: Your Machine Learning and Data Science Community. (s.f.). Obtenido de https://www.kaggle.com/

Lewis, R. (1998). *Why Pattern Search Works.* Virginia: Institute for Computer Applications in Science and Engineering. Obtenido de https://apps.dtic.mil/sti/pdfs/ADA359194.pdf

Princewill, E. (2021). *Analysis and Performance Evaluation of Selected Pattern Matching Algorithms.* Oghara, Delta State, Nigeria: Department of mathematics & Computer Science, Western Delta University.

Trivedi, U. (02 de Agosto de 2021). *Z algorithm (Linear time pattern searching Algorithm)*. Obtenido de Geeks For Geeks: https://www.geeksforgeeks.org/z-algorithm-linear-time-pattern-searching-algorithm/

Vonteru, K. (03 de Agosto de 2019). *Generating short length descriptions of news articles.* Obtenido de Kaggle: https://www.kaggle.com/sunnysai12345/news-summary