Rendimiento de la búsqueda de palabras mediante 3 algoritmos de búsqueda distintos de Python.

Vladimir Osvaldo Curiel Ovalles
Escuela de Ingeniería en Computación y Telecomunicaciones
Pontificia Universidad Católica Madre y Maestra
Santiago de los Caballeros, República Dominicana
VOCO0001@ce.pucmm.edu.do

Armando José González López
Escuela de Ingeniería en Computación y Telecomunicaciones
Pontificia Universidad Católica Madre y Maestra
Santiago de los Caballeros, República Dominicana
AJGL0001@ce.pucmm.edu.do

Resumen

En esta investigación se quiere comparar el rendimiento en términos de tiempo de tres métodos de búsqueda de palabras de la librería re de Python: findall, search y finditer. Utilizando datos cuantitativos y partiendo de un muestreo estratificado, se tomaron distintos textos arbitrados de una determinada categoría. Se analizaron 374 resúmenes extraídos de Scopus, se obtuvieron las 10 palabras más repetidas, y se midió el tiempo utilizando dos métodos de medición de tiempo distintos: timeit y perf_counter. Este proceso se repitió 10,000 veces para obtener un promedio de tiempo fiable, lo que dio un total de 224,400,000 iteraciones. Los resultados revelan que finditer es el algoritmo más rápido para realizar la tarea, con una diferencia del 93.51% de eficiencia según los tiempos obtenidos al medir.

Abstract

In this research, we aim to compare the performance in terms of time of three search methods from Python's re library: findall, search, and finditer. Using quantitative data and based on stratified sampling, various peer-reviewed texts from a specific category were analyzed. A total of 374 abstracts extracted from Scopus were used to identify the 10 most repeated words, and the time taken by each method was measured using two different timing methods: timeit and perf_counter. This process was repeated 10,000 times to obtain a reliable average time, resulting in a total of 224,400,000 iterations. The results reveal that finditer is the fastest algorithm for the task, with a 93.51% efficiency difference according to the measured times.

Palabras Clave

- Rendimiento de algoritmos de búsqueda
- Expresiones regulares en Python
- Tiempo de ejecución

Introducción

La búsqueda de palabras dentro de textos es una tarea fundamental que podemos utilizar en diversas áreas de las ciencias de la computación y la informática en general, como puede ser en el procesamiento de lenguaje natural, que consiste en la utilización de un lenguaje natural para comunicarnos con la computadora, debiendo ésta entender las oraciones que le sean proporcionadas [1], la minería de datos, que se trata de descubrir tendencias, desviaciones, comportamientos atípicos, patrones y trayectorias ocultas [2], entre otras áreas. Evaluar y comparar el rendimiento de estos algoritmos es importante debido a lo útil que puede ser conocer cuál es el algoritmo más rápido al momento de trabajar con diversos volúmenes de datos y sobre todo con grandes cantidades de datos. La investigación parte de 3 algoritmos de búsqueda de palabras, findall, search y finditer, que provienen de la librería re de Python para trabajar expresiones regulares, que sirven para representar lenguajes regulares y tienen diferentes aplicaciones [3].

Se investigo acerca de estudios específicos de comparación de algoritmos de búsquedas de palabras, a pesar de dicha búsqueda, no se obtuvieron resultados de un estudio exacto más allá de discusiones en foros de programación y/o artículos puntuales. Debido a esto, se tiene como objetivo principal comparar el rendimiento de 3 algoritmos de búsqueda de palabras, los cuales son findall, finditer y search, los cuales serán medidos en términos de tiempo, siendo el mejor que el que encuentre todas las palabras en el menor tiempo. Los algoritmos de búsqueda de palabras provenientes de la librería re, tienen un objetivo en común pero cada uno tiene un funcionamiento interno distinto lo permite que los tiempos de búsqueda sean distintos.

Se mencionó anteriormente que cada uno de los algoritmos tiene un funcionamiento distinto, e incluso para realizar las pruebas, cada uno puede necesitar unos prerrequisitos específicos para poder llevar a cabo la comparación de manera precisa. Es necesario explicar cómo funcionan estos algoritmos, los retornos que tienen y los parámetros que estos reciben, y si tienen algún prerrequisito para que las medidas sean precisas. Iniciando por el objeto Match, es importante entender este objeto, ya que da cabida a entender las diferencias de rendimientos entre los algoritmos, el objeto Match es aquel que es devuelto luego de que una búsqueda haya sido exitosa, pudiendo ser un arreglo de strings o bytes de matching [4]. Search, escanea a través del texto buscando la primera coincidencia de la expresión regular, retornando su match correspondiente con el String de coincidencia [5]. Findall, busca todas las ocurrencias del patrón en el texto, retornando su match correspondiente con los String de coincidencia [6]. Finditer por su parte, puede dar más información que solo las palabras, hace el mismo trabajo que Findall, pero retorna la instancia del objeto Match simplemente con las posiciones en vez de los Strings de coincidencia [7].

Los resultados de esta investigación pretenden dar respuesta a la hipótesis planteada sobre cuál de los 3 algoritmos es más eficiente en términos de tiempo. Como fue mencionado anteriormente, conocer cuál es, puede ayudar obtener resultados más rápidos cuando se está trabajando con grandes volúmenes de datos, ya que tratar con volúmenes pequeños de datos, a pesar de que exista una diferencia significativa, la diferencia de tiempo entre algoritmo puede ser irrelevante y se prefiera usar cualquiera de los 3. Asimismo, se podrá dar bases a investigaciones futuras más robustas acerca de esta comparación con más algoritmos y mayor volumen de datos, de momento, podremos dar una conclusión a la hipótesis tratada y tener noción de cuál algoritmo es el más eficiente.

Desarrollo

El propósito principal en este apartado es dar a conocer en detalle cual es la metodología utilizada al momento de llevar a cabo la investigación, los pasos a seguir, instrumentos de medida y los procesos que se llevaron a cabo para aplicar cada instrumento en los algoritmos. La investigación tiene un enfoque cuantitativo, a esto se hace referencia a que parte de un punto estadístico, teniendo una población de datos a la cual se le obtuvo una muestra que fue tratada por estratos, por un procedimiento llamado muestreo aleatorio estratificado implica la selección al azar de una muestra *dentro* de cada estrato [8], permitiendo que cada categoría de datos se encuentre de manera homogénea y otorgando la misma probabilidad de ser elegido.

La población presente en esta investigación está formada por texto arbitrados en el idioma inglés, dicha población está centrada en textos cuya categoría principal es las Ciencias de la Computación, manteniendo así un vocabulario homogéneo y tener una serie de palabras comunes a lo largo de todos los textos y poder probar efectivamente la búsqueda de palabras. La muestra con la que se trabajó parte de 3 categorías de textos, editoriales, capítulos de libros y reseñas, extraídos de Scopus, una base de datos de resúmenes y citas de literatura revisada por pares, que incluye revistas científicas, libros y actas de conferencias [9], donde la población de textos encontrados bajos las 3 categorías mostradas fue de 12,773 textos presentes en Scopus, de los cuales se extrajeron 374 muestras en totales. Se dividió en 3 estratos, uno para cada categoría mencionada previamente, editoriales, capítulos de libros y reseñas y se repartió la cantidad de textos de cada de manera representativa de la población, resultando eso en 153 de editorial, 122 de capítulos de libro y 99 de reseñas. De cada una de las muestras se trabajó utilizando el resumen, que en promedio cada uno tiene 105 palabras, de las cuales se extrajeron las 10 más comunes para tener un listado de palabras para probar.

Los instrumentos de medida seleccionados para realizar dichas pruebas son 2 algoritmos para medir el tiempo perf_counter() y timeit. Perf_counter() devuelve el valor (en segundos fraccionarios) de un contador de rendimiento, es decir, un reloj con la máxima resolución disponible para medir una duración corta [10]. Por otro lado, timeit crea una instancia de un timer, recibe una función a repetir y el número de

veces que se tiene ejecutar, este otorgara el promedio de todas las ejecuciones [11]. Estos métodos nos proporcionan ambos una manera eficiente de obtener el tiempo que tardan los algoritmos de búsquedas de palabra en obtener los resultados y poder con ese tiempo obtener cual es más eficiente y en qué condiciones se cumple el resultado.

La forma en la que los algoritmos fueron medidos es la siguiente, cada uno de los 3 algoritmos fue medido con los 2 instrumentos, para esto se utilizaron los 374 con las 10 palabras más repetidas y cada uno fue evaluado un total de 10,000 (3 * 2 *374 * 10 * 10000), esto nos da un resultado de 224,400,000 iteraciones totales que fueron ejecutadas entre los diferentes métodos con los diversos instrumentos, estas permiten hallar tiempos promedios de ejecución fiables. Tratando los datos de esta forma, hace posible obtener una gama de resultados variables, ya sea resultados por algoritmo con instrumento específico, resultados por palabras, resultados por palabras con un instrumento específico, en general, se puede obtener una variedad de resultados y observar la veracidad de los resultados.

Los resultados obtenidos se presentan en formato de gráficos de las medidas de tiempo promedio y sus distribuciones normales, la cual describe de manera aproximada muchos fenómenos que ocurren en la naturaleza, la industria y la investigación [12], para validar la veracidad de los tiempos promedios. Estos gráficos permiten visualizar de manera clara cuál es el resultado. A pesar de tener una gran gama de resultados, todos arrojaron un mismo resultado, bajo cualquiera palabra, texto e instrumento de medida, el algoritmo más rápido para realizar la tarea, con una diferencia del 93.51% de eficiencia según los tiempos obtenidos al medir es finditer. Se puede observar de manera gráfica las diferencias que existen entre los 3 algoritmos, donde se recalca que, a menor altura mejor resultado.

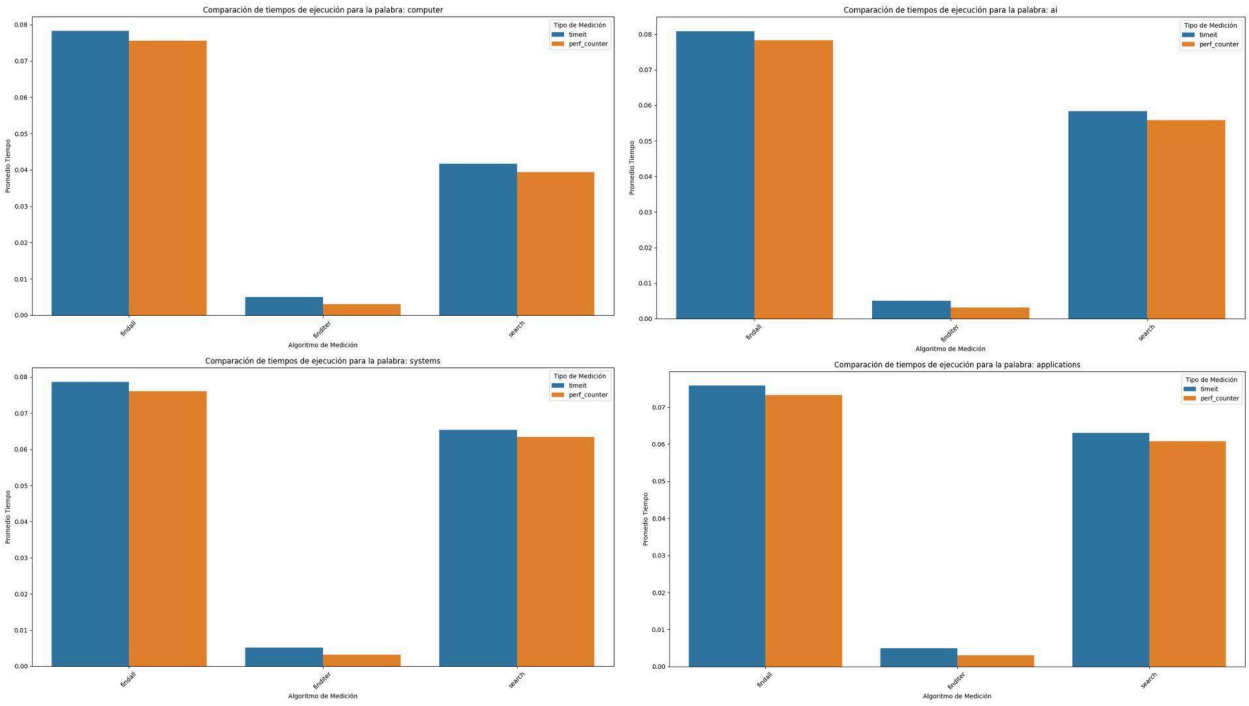

Figura 1: Gráficos de las palabras ‘computer’, ‘ai’, ‘systems’, ‘applications’.

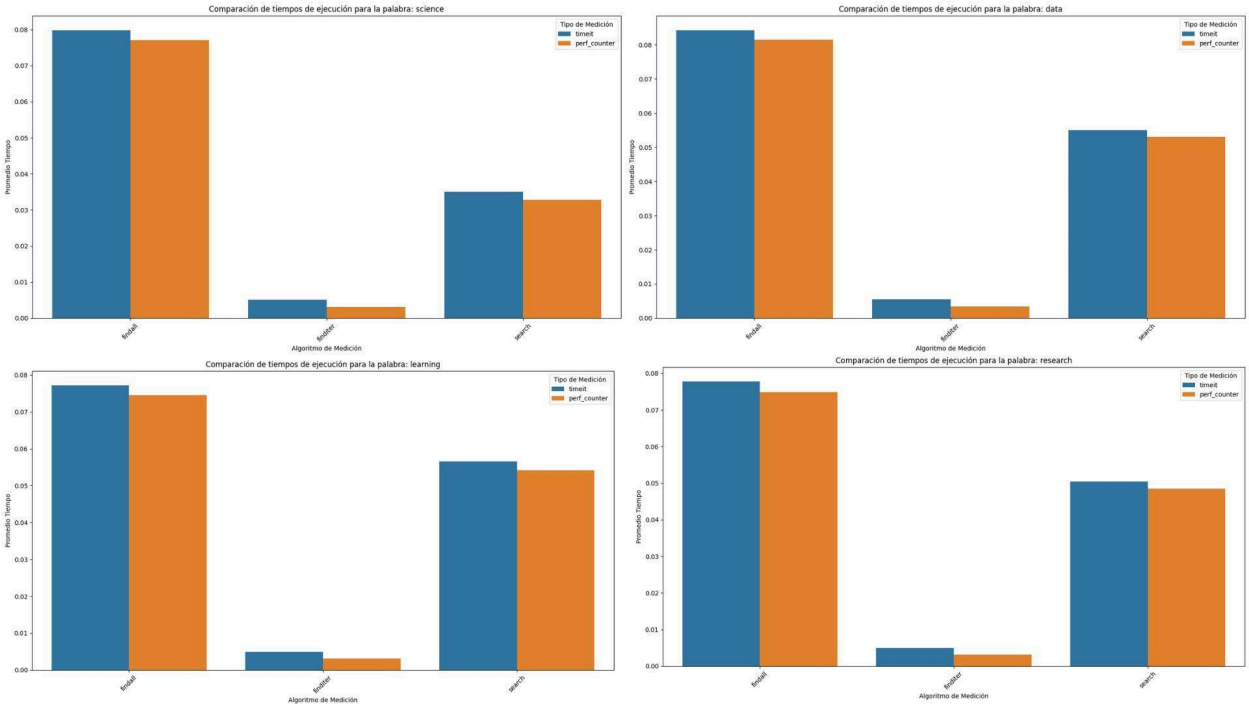

Figura 2: Gráficos de las palabras ‘science, ‘data, ‘learning’, ‘research’.

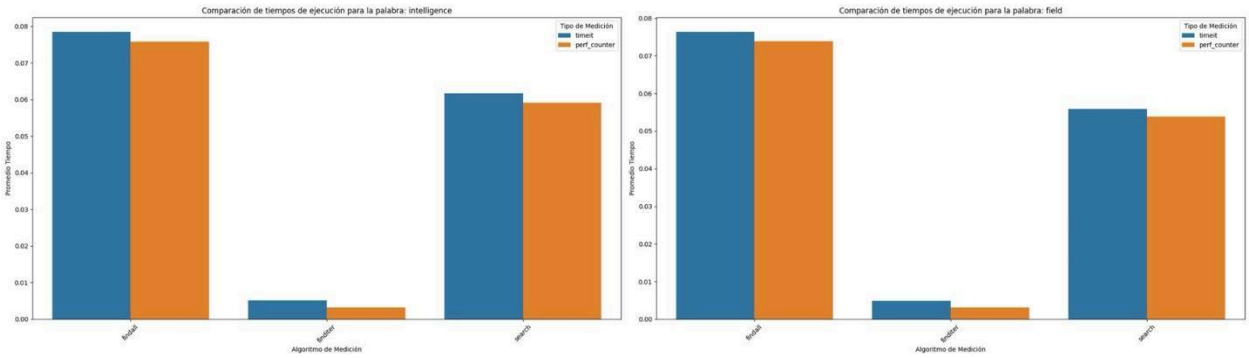

Figura 3: Gráficos de las palabras ‘intelligence', ‘field’.

Se puede observar que bajo cada uno de los gráficos mostrados de cada una de las palabras que fueron utilizadas para hacer las pruebas, los resultados son similares para cada algoritmo con cada instrumento de medición. Del mismo modo, al observar los datos en promedio de todas las palabras, se puede apreciar un gráfico muy similar al de cada una de las palabras presentadas, dada la consistencia de los resultados obtenidos.

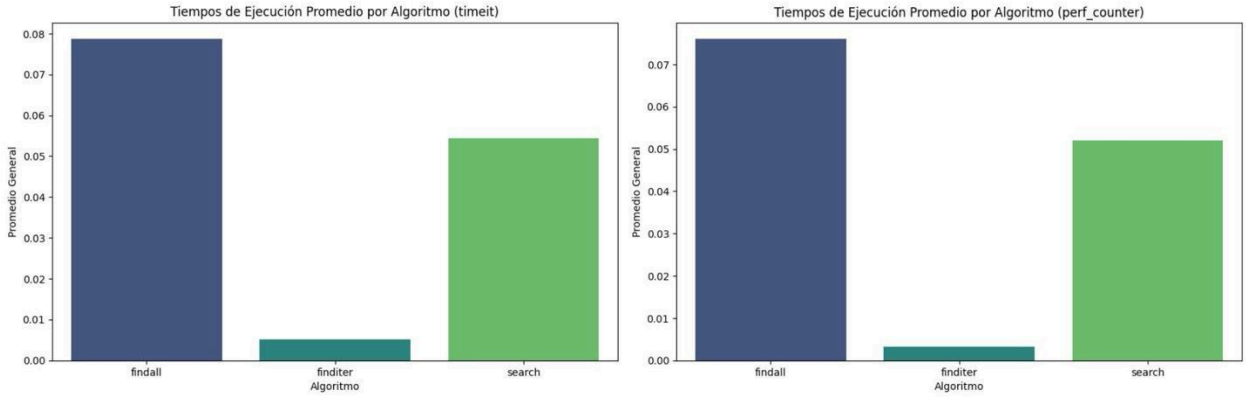

Figura 4: Gráficos de los tiempos promedios de cada algoritmo con ambos instrumentos de medida.

Asimismo, se presenta la distribución normal de los datos de los tiempos promedio de cada algoritmo con promedio de ambos instrumentos, dónde se muestra cómo están distribuidos los resultados obtenidos y valida los resultados de los promedios obtenidos.

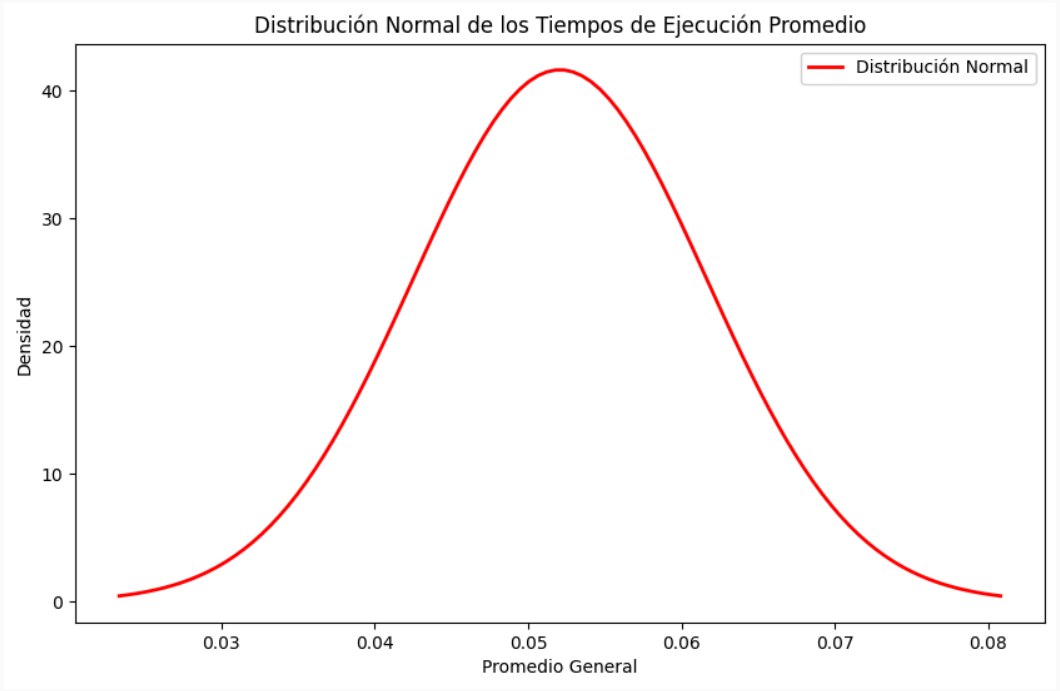

Figura 5: Gráficos de la distribución normal de los tiempos promedio de cada uno de los algoritmos (promedio de ambos instrumentos)

	Algoritmo	Tipo de Medición	Promedio General
0	findall	timeit	0.078778
1	findall	perf_counter	0.076087
2	finditer	timeit	0.005067
3	finditer	perf_counter	0.003154
4	search	timeit	0.054317
5	search	perf_counter	0.052074

De manera numérica, se pueden presentar los datos y tener los resultados precisos de cada uno con ambos instrumentos de medida, dónde se aprecia la misma consistencia de datos en los resultados, los cuales pueden validar el resultado de que, ‘finditer’ es el algoritmo más eficiente para buscar palabras. Una de las posibles razones ante tal resultado, puede tener origen en la forma en la que los resultados de todos los métodos son almacenados. Al comienzo de esta investigación se explicó acerca del objeto

Match, que es el objeto de retorno de los métodos de búsqueda de palabras el cual puede ser un arreglo de caracteres (las palabras encontradas) o puede ser un arreglo de bytes, dónde se colocan las posiciones exclusivamente. Finditer es el único método que guarda las posiciones en bytes y no necesita agregar las palabras al arreglo Match como caracteres para ser presentado en una salida inmediata, debido a lo cual se hace el acercamiento a una posible razón de los resultados obtenidos. Asimismo, se entiende que el resultado de Finditer no son las palabras como tal si no sus índices, estás se pueden obtener fácilmente iterando sobre el iterador que devuelve, a pesar de que esto puede agregar un tiempo extra al conjunto total de los que es una búsqueda de palabras, se realizaron alguna pruebas menores y se observó que el tiempo que se toma para iterar sobre los índices no aporta un tiempo significativo a los resultados.

Conclusiones

En este estudio se ha evaluado y comparado el rendimiento en términos de tiempo de tres métodos de búsqueda de palabras de la librería re de Python: findall, search y finditer. Con un enfoque cuantitativo basado en un muestreo estratificado, se analizaron 374 resúmenes de Scopus para identificar las 10 palabras más repetidas y se midió el tiempo de ejecución utilizando timeit y perf_counter. Este proceso se repitió 10,000 veces, dando un total de 224,400,000 iteraciones. Los resultados revelan que finditer es el algoritmo más eficiente, mostrando una diferencia del 93.51% en la eficiencia según los tiempos obtenidos.

La importancia de estos resultados es la utilidad práctica de elegir el algoritmo más rápido cuando se trabaja con grandes volúmenes de datos. En escenarios donde el tiempo de procesamiento es crucial, finditer se destaca como la opción preferida. A pesar de que findall y search también cumplen con la tarea de búsqueda, sus tiempos de ejecución son considerablemente mayores en comparación con finditer, lo que los hace menos adecuados para aplicaciones donde se requiere alta eficiencia temporal.

Para futuros trabajos, estos resultados pueden servir como base para explorar mejoras adicionales en los algoritmos de búsqueda de palabras. Una posible área de investigación podría ser la optimización de estos métodos para su uso en sistemas de procesamiento de lenguaje natural (NLP) más avanzados, donde la rapidez y precisión son esenciales. Además, se puede investigar cómo estos algoritmos se comportan con datos de diferentes naturalezas y tamaños, incluyendo textos en otros idiomas o documentos técnicos con terminología específica. Investigaciones futuras podría ser la integración de estos algoritmos en aplicaciones de minería de datos más complejas, donde no solo se busca la rapidez, sino también la capacidad de manejar grandes cantidades de datos con precisión. La combinación de finditer con técnicas de paralelización y procesamiento distribuido podría proporcionar mejoras significativas en términos de rendimiento y escalabilidad.

Bibliografía

[1]: *Vásquez, Augusto Cortez, et al. "Procesamiento de Lenguaje Natural." Revista de Investigación de Sistemas e Informática, vol. 6, no. 2, 2009, pp. 45–54.* *https://revistasinvestigacion.unmsm.edu.pe/index.php/sistem/article/view/5923/5121*

[2]: Martínez, B. Beltrán. "Minería de Datos." *Cómo Hallar Una Aguja En Un Pajar. Ingenierías*, vol. 14, no. 53, 2001, pp. 53–66. https://www.cs.buap.mx/~bbeltran/NotasMD.pdf

[3]: *Farías, Gustavo. Expresiones Regulares En GNU/Linux. 2023.* *https://repositorio.cfe.edu.uy/handle/123456789/2286*

[4]: *re Match — Regular expression operations*. (n.d.). Python Documentation. https://docs.python.org/3/library/re.html#re.Match

[5]: *re Search — Regular expression operations*. (n.d.). Python Documentation. https://docs.python.org/3/library/re.html#re.Search

[6]: *re Findall — Regular expression operations*. (n.d.). Python Documentation. https://docs.python.org/3/library/re.html#re.finding-all-adverbs

[7]: *re Finditer — Regular expression operations*. (n.d.). Python Documentation. https://docs.python.org/3/library/re.html#re.finding-all-adverbs-and-their-positions

[8]: Muestreo estratificado, Walpole, R. E. (2012). *PROBABILIDAD y ESTADISTICA PARA INGENIEROS y CIENCIAS, 9ED*. (p.8)

[9]: Elsevier. (n.d.). *What is Scopus Preview? - Scopus: Access and use Support Center*. https://service.elsevier.com/app/answers/detail/a_id/15534/supporthub/scopus/#tips

[10]: *time – perf_counter() — Time access and conversions*. (n.d.). Python Documentation. https://docs.python.org/3/library/time.html#time.perf_counter

[11]: *timeit — Measure execution time of small code snippets*. (n.d.). Python Documentation. https://docs.python.org/es/3/library/timeit.html#timeit.timeit

[12]: Muestreo estratificado, Walpole, R. E. (2012). *PROBABILIDAD y ESTADISTICA PARA INGENIEROS y CIENCIAS, 9ED*. (p.172)