ANALISIS DE GRAFICAS COMPLEJIDAD ALGORITMICA(ALIEN VS DEPREDADOR) (Noviembre de 2023)

Jerónimo Velásquez Martínez, Isabella Montoya Castellanos, Wilson Santiago Carvajal, Sara Medina Molina. Estudiantes de Ingeniería de Sistemas de la Universidad Medellín

Resumen - La complejidad algorítmica y el análisis de Big O representan piedras angulares en el desarrollo de software, permitiendo la evaluación cuantitativa del rendimiento y la eficiencia de los algoritmos. Este artículo profundiza en la complejidad algorítmica, explorando su estructura conceptual y su aplicación en diversas áreas de la ingeniería de software. Se examinan detalladamente las estrategias para medir la complejidad temporal y espacial, junto con su influencia en la eficiencia computacional. A través de ejemplos prácticos y casos de estudio, se ilustran las técnicas para mejorar el rendimiento de los algoritmos y se abordan los desafíos asociados, ofreciendo una guía esencial para desarrolladores y profesionales en busca de optimización y eficacia en sus implementaciones.

Palabras Clave- Eficiencia Algorítmica, Análisis de Rendimiento, Complejidad Temporal, Complejidad Espacial

I. INTRODUCCIÓN

El Big O y la complejidad algorítmica introducen un lenguaje cuantitativo para medir la eficiencia temporal y espacial de los algoritmos. Al explorar este campo fascinante, este artículo se sumerge en la esencia de Big O y la complejidad algorítmica, examinando su evolución, fundamentos y aplicaciones prácticas. Al comprender la importancia de estos conceptos y su papel central en el diseño de algoritmos eficientes, los desarrolladores se equipararán con las herramientas necesarias para afrontar los desafíos computacionales con destreza y precisión en el cambiante panorama del desarrollo de software.

Documento recibido el 9 de octubre de 2001. (Anote la fecha en que usted presentó su documento para su revisión.) Este trabajo fue apoyado en parte por los U.S. Department of Commerce under Grant S123456 (reconocimiento al patrocinador y apoyo financiero va aquí). los títulos del Documento deben ser escritos en letras mayúsculas y minúsculas, no todas las mayúsculas. Evite escribir fórmulas extensas con subíndices en el título; Utilice Fórmulas cortas que identifiquen los elementos (por ejemplo, "Nd-Fe-B"). No escriba "(invitados)" en el título. Escriba los Nombres completos de los autores en el campo autor, pero no es necesario. Ponga un espacio entre los autores.

F. A. Author is with the National Institute of Standards and Technology, Boulder, CO 80305 USA (corresponding author to provide phone: 303-555-5555; fax: 303-555-5555; e-mail: author@ boulder.nist.gov).

S. B. Author, Jr., was with Rice University, Houston, TX 77005 USA. He is now with the Department of Physics, Colorado State University, Fort Collins, CO 80523 USA (e-mail: author@lamar. colostate.edu).

T. C. Author is with the Electrical Engineering Department, University of Colorado, Boulder, CO 80309 USA, on leave from the National Research Institute for Metals, Tsukuba, Japan (e-mail: author@nrim.go.jp).

A. *Terminos del Big O*

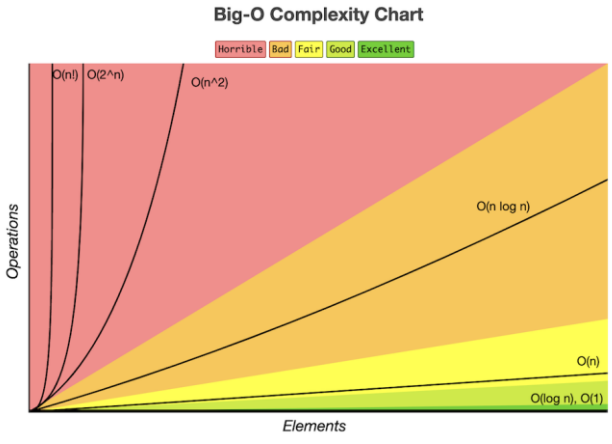
O(1) - *Constante:* Indica que el tiempo de ejecución del algoritmo no cambia independientemente del tamaño de la entrada. Es una operación que toma una cantidad fija de tiempo para completarse, como acceder a un elemento en un array.

O(n) - *Lineal:* El tiempo de ejecución crece de manera proporcional al tamaño de la entrada. Por ejemplo, recorrer una lista una vez.

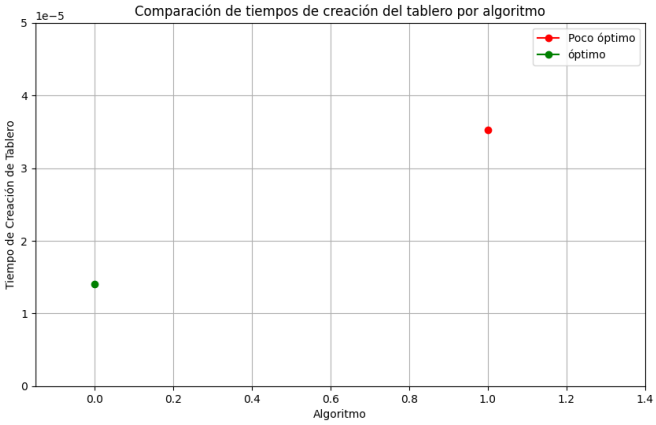
O(log n) - *Logarítmica:* A medida que aumenta el tamaño de la entrada, el tiempo de ejecución crece, pero de manera logarítmica. Al dividir iterativamente el conjunto de datos, el algoritmo puede reducir el tamaño del problema con cada iteración, como en la búsqueda binaria.

O(n^2) - *Cuadrática:* El tiempo de ejecución crece proporcional al cuadrado del tamaño de la entrada. Se observa en algoritmos con bucles anidados, como al comparar cada elemento en una lista con cada otro elemento.

O(2^n) - *Exponencial:* El tiempo de ejecución aumenta exponencialmente con el tamaño de la entrada. Es común en algoritmos recursivos que generan una cantidad exponencial de subproblemas, como la solución de fuerza bruta para algunos problemas de combinaciones o permutaciones.



II. ANALISIS DE TIEMPO DE EJECUCIÓN



Realizando el análisis, visualizamos que el tiempo de ejecución de la creación del tablero con una lista enlazada, demora mas tiempo que una matriz.

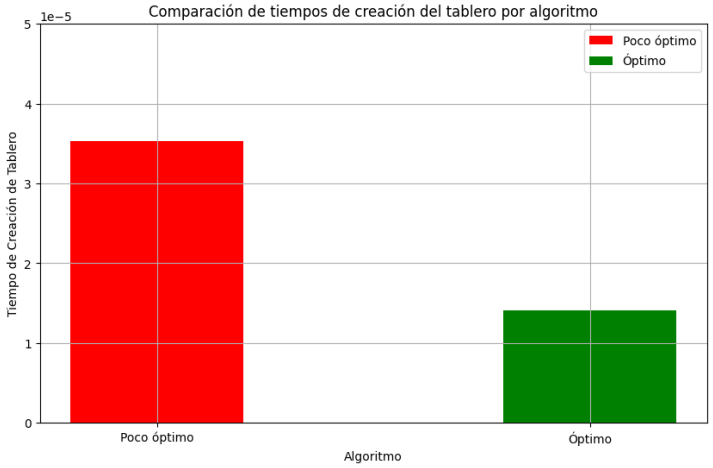
La diferencia en el tiempo de ejecución entre la creación de un tablero usando una lista enlazada y una matriz puede deberse a la forma en que se almacenan los datos y se accede a ellos en la memoria.

Lista enlazada: Cada nodo en la lista enlazada contiene una referencia al siguiente nodo, lo que significa que los datos no están necesariamente contiguos en la memoria. Al crear una matriz usando una lista enlazada, se requiere crear múltiples nodos para representar las filas y columnas, lo que implica una mayor cantidad de asignaciones de memoria y operaciones de enlace.

Además, acceder a elementos específicos en una lista enlazada implica recorrer los nodos secuencialmente desde el principio hasta la posición deseada. En una matriz representada por lista enlazada, esto puede requerir más operaciones de acceso y navegación para llegar a una celda específica, especialmente en matrices grandes.

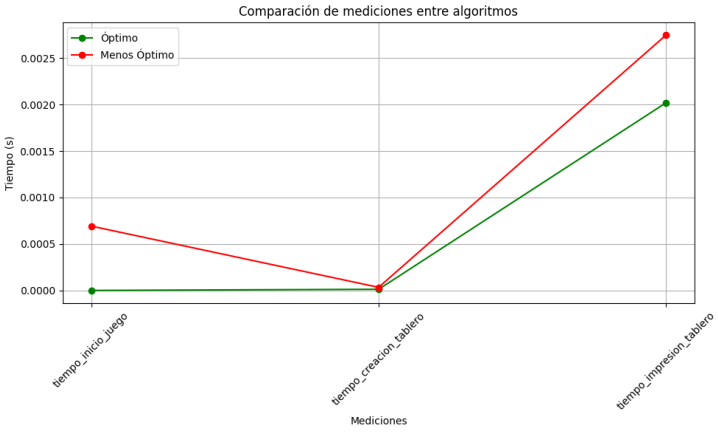
Matriz: En comparación, una matriz almacena los elementos en una disposición bidimensional contigua en la memoria. Esto permite un acceso más directo a los elementos utilizando índices de fila y columna. La creación de una matriz implica asignar un bloque de memoria contiguo para todos los elementos, lo que puede ser más eficiente en términos de acceso y manipulación de datos, especialmente para operaciones que implican el acceso aleatorio a celdas específicas.

Se puede observar mucho más claro en un gráfico a barras.



En resumen, la lista enlazada puede requerir más tiempo de ejecución debido a las operaciones adicionales de enlace y acceso secuencial, mientras que una matriz puede ser más eficiente en el acceso directo a los elementos debido a su disposición contigua en memoria.

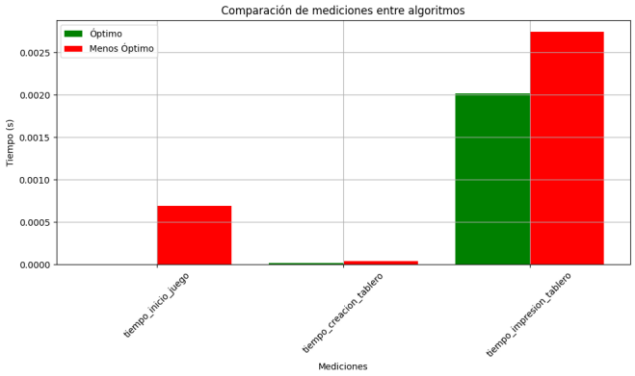
A. Análisis de Tiempo General



El objetivo era determinar cuál de estos enfoques ofrecía un rendimiento más óptimo en términos de tiempo de ejecución en relación con el tamaño de la entrada.

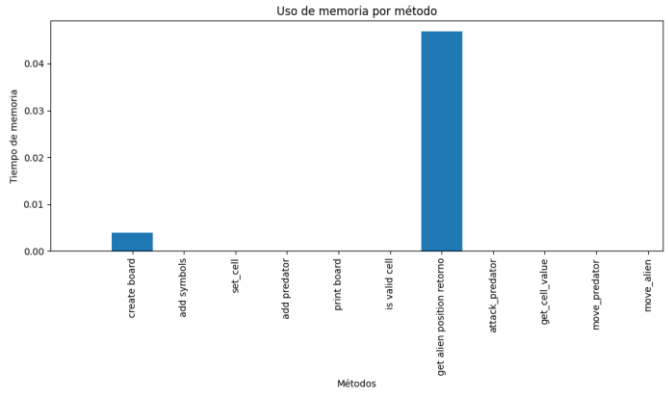
La línea que representa el algoritmo basado en matrices se posicionó consistentemente por debajo de la línea correspondiente al algoritmo basado en listas enlazadas en todo el espectro de tamaños de entrada evaluados.

Este hallazgo significativo indica que, sin importar el tamaño de la entrada, el algoritmo de matriz exhibió tiempos de ejecución menores en comparación con su contraparte basada en listas enlazadas.

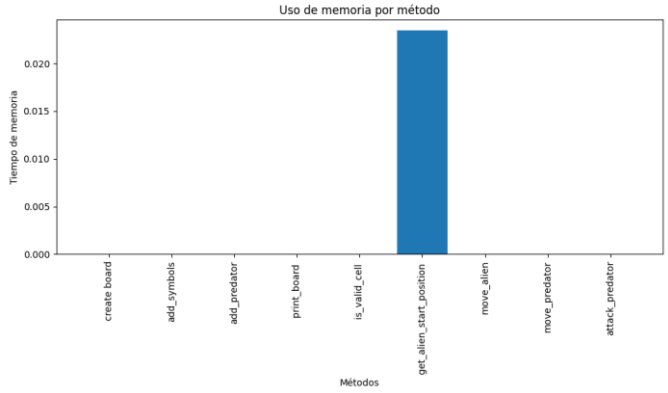


Esto sugiere de manera concluyente que, para este escenario particular de análisis y operaciones, el enfoque algorítmico basado en matrices se destacó como más eficiente. Su capacidad para realizar operaciones con tiempos de ejecución más reducidos, incluso a medida que aumentaba el tamaño de la entrada, lo posicionó como la opción preferida en términos de optimización temporal.

III. ANALISIS DE TIEMPO EN CADA METODO



Visualizando los metodos de ejecución en la creación del tablero, con una lista enlazada es mucho mayor.



Visualizandolo con una matriz, vemos que el tiempo de ejecución es mas minimo al momento de la creación del tablero.

IV. CONCLUSIONES

Rendimiento y estructuras de datos: La selección de la estructura de datos adecuada puede impactar significativamente el rendimiento de un algoritmo. En este caso, la matriz demostró ser más eficiente en términos de tiempo de ejecución en comparación con la lista enlazada para las operaciones evaluadas.

Tamaño de entrada y escalabilidad: A medida que el tamaño de la entrada aumenta, la diferencia en el rendimiento entre las estructuras de datos puede volverse más evidente. La matriz mantuvo un rendimiento constante y superior a medida que crecía el tamaño de la entrada, lo que sugiere una mejor escalabilidad en comparación con la lista enlazada.

Optimización y mejora continua: Este análisis subraya la importancia de la evaluación continua y la optimización de algoritmos y estructuras de datos para mejorar el rendimiento. La exploración de diferentes enfoques puede llevar a mejoras significativas en la eficiencia y el tiempo de ejecución.

V. REFERENCIAS

[1] https://www.campusmvp.es/recursos/post/Rendimiento-de-algoritmos-y-notacion-Big-O.aspx

[2] https://www2.infor.uva.es/~jvalvarez/docencia/tema5.pdf

[3] https://www.freecodecamp.org/espanol/news/explicacion-de-la-notacion-big-o-con-ejemplo/