Instituto Tecnológico y de Estudios Superiores de Monterrey

Arturo Sanchez Rodriguez - A01275427

Programación de estructura de datos y algoritmos fundamentales (Gpo 850)

reflexFinTC1031

13 de Junio del 2023

Introducción -

Reflexionar sobre el curso de estructuras de datos es adentrarse en un fascinante mundo de algoritmos y organización de información. A lo largo de este período académico, hemos explorado una amplia variedad de estructuras, desde algoritmos de ordenamiento hasta tablas hash, cada una de las cuales desempeña un papel crucial en la optimización de procesos y la gestión eficiente de datos.

Una de las primeras lecciones que aprendimos fue que no existe una estructura de datos universalmente perfecta. Cada una de ellas tiene fortalezas y debilidades que las hacen más adecuadas para ciertos escenarios. Por ejemplo, los algoritmos de ordenamiento son ideales cuando se requiere organizar grandes conjuntos de datos, mientras que las estructuras de datos lineales, como las listas enlazadas o los vectores, se destacan en la gestión de elementos secuenciales.

El estudio de los árboles de búsqueda binaria nos abrió los ojos a la potencia de la búsqueda rápida y eficiente en conjuntos de datos ordenados. La capacidad de dividir y conquistar, de manera recursiva, a través de las ramas de un árbol nos permitió obtener resultados en tiempos de ejecución sorprendentemente bajos. Asimismo, los grafos nos mostraron cómo las relaciones complejas pueden ser representadas y analizadas mediante nodos interconectados, ofreciendo soluciones innovadoras a problemas de conectividad y optimización.

Sin embargo, quizás una de las mayores reflexiones que nos acompañará más allá de este curso es la importancia de considerar tanto la complejidad computacional como el costo computacional al elegir una estructura de datos. No se trata sólo de implementar una estructura porque suena interesante o está de moda, sino de evaluar minuciosamente su desempeño en diferentes situaciones. Es crucial analizar los escenarios posibles, las operaciones más frecuentes y las restricciones de recursos, para así seleccionar la estructura más eficiente y evitar ineficiencias innecesarias.

En resumen, este curso nos ha brindado una valiosa perspectiva sobre la diversidad y utilidad de las estructuras de datos. Hemos aprendido que cada una de ellas es un valioso instrumento en nuestro arsenal de programación, capaz de mejorar significativamente el rendimiento de nuestros algoritmos y aplicaciones. La clave reside en comprender sus características y

aplicaciones, así como en saber elegir sabiamente la estructura adecuada para cada situación. A medida que avanzamos en nuestra formación, esta reflexión nos acompañará y nos recordará la importancia de la eficiencia y la optimización en el mundo de la programación y el manejo de datos.

Algoritmos de Ordenamiento

En el campo de la programación, existen diversos algoritmos de ordenamiento que pueden ser utilizados para ordenar conjuntos de datos. Algunos de los algoritmos más comunes son bubble sort, merge sort, quick sort e insertion sort. Sin embargo, la eficiencia de cada algoritmo puede variar según el tamaño de los datos a ordenar y su estructura. En este sentido, es importante analizar detalladamente las características de cada algoritmo y su complejidad temporal.

En primer lugar, se puede observar que algunos algoritmos presentan una complejidad mayor que otros. Por ejemplo, el bubble sort e insertion sort tienen una complejidad cuadrática, lo que significa que su desempeño empeora considerablemente a medida que el número de datos aumenta. Estos algoritmos son más adecuados para conjuntos de datos pequeños o ya casi ordenados, ya que su eficiencia se ve afectada en situaciones más complejas.

Por otro lado, el quicksort y mergesort son considerados algoritmos más eficientes en términos de tiempo de ejecución promedio. El quicksort utiliza una estrategia de divide y conquista, seleccionando un elemento llamado "pivote" y colocándolo en su posición final, mientras divide el conjunto de datos en dos subconjuntos que se ordenan por separado. La elección adecuada del pivote es crucial para un buen rendimiento del quicksort. Por otro lado, el merge sort también utiliza la estrategia de divide y conquista, dividiendo el conjunto de datos en subconjuntos más pequeños, los cuales se ordenan de forma recursiva y luego se combinan en un solo conjunto ordenado.

Ambos algoritmos, quicksort y merge sort, presentan una complejidad temporal promedio de O(N log N), lo que los hace más eficientes en comparación con bubble sort e insertion sort. Sin embargo, es importante tener en cuenta que la elección del algoritmo de ordenamiento adecuado depende de las características específicas del conjunto de datos y del contexto en el que se está trabajando.

Para mejorar aún más la eficiencia de los algoritmos de ordenamiento, se pueden implementar técnicas adicionales. Por ejemplo, en el caso del quicksort, una selección cuidadosa del pivote puede conducir a una mejora significativa en el rendimiento. Además, se pueden combinar diferentes técnicas de ordenamiento en algoritmos híbridos, adaptándose según el caso específico. Estas técnicas permiten aprovechar las fortalezas de varios algoritmos de ordenamiento y adaptar su uso según las necesidades y características de los conjuntos de datos a ordenar.

Por último, la elección del algoritmo de ordenamiento adecuado es fundamental para lograr una eficiencia óptima en la tarea de ordenar conjuntos de datos. Al considerar factores como la complejidad temporal, el tamaño del conjunto de datos y su estructura, es posible seleccionar el algoritmo más apropiado. Asimismo, la implementación de técnicas adicionales, como la selección adecuada del pivote o la combinación de diferentes algoritmos, puede contribuir a mejorar aún más la eficiencia del proceso de ordenamiento.

Temas vistos en clase

Algoritmos de ordenamiento,

Dentro del ámbito de la clasificación de datos, podemos encontrar una variedad de algoritmos de ordenamiento, tales como bubble sort, merge sort, quick sort e insertion sort. La eficiencia de cada algoritmo puede variar en función del tamaño y la estructura de los datos a ordenar. Se ha observado que algunos algoritmos presentan una complejidad temporal mayor que otros, lo que implica un rendimiento inferior a medida que aumenta la cantidad de datos.

En este sentido, se destaca la eficiencia del quicksort y mergesort, los cuales poseen una complejidad promedio, a diferencia del bubble sort e insertion sort, que tienen una complejidad cuadrática. No obstante, es posible mejorar la eficiencia de estos algoritmos mediante la implementación de estrategias de selección de pivotes en el caso del quicksort, o incluso combinando diferentes técnicas de ordenamiento según sea necesario.

Estructuras de datos lineales,

Dentro del estudio de las estructuras de datos lineales, se han explorado las listas enlazadas (linkedlist) y las listas doblemente enlazadas (doubly linked list). Si bien las listas enlazadas son eficientes en términos de inserción y eliminación de datos, presentan una desventaja en cuanto a la búsqueda o acceso a un dato específico, ya que se requiere recorrer toda la lista. Por otro lado, las estructuras de datos vectoriales, como los arrays, ofrecen un acceso más directo a los datos. Para mejorar la eficiencia de las listas enlazadas, es posible considerar la implementación de vectores dinámicos, los cuales brindan un acceso más eficiente a los elementos almacenados.

Árboles de búsqueda binaria,

Los árboles de búsqueda binaria constituyen estructuras de datos eficientes para el acceso y la búsqueda de elementos. En general, presentan un rendimiento óptimo en diversas operaciones, como la búsqueda, inserción y eliminación de datos, gracias a su complejidad computacional logarítmica. Sin embargo, la eficiencia de estos árboles está estrechamente relacionada con su balanceo. Para mejorar su eficiencia, es posible utilizar árboles AVL, los cuales mantienen un equilibrio de altura y, por lo tanto, garantizan un rendimiento óptimo en las operaciones realizadas sobre ellos.

Grafos,

Los grafos representan estructuras de datos altamente versátiles, que nos permiten visualizar y comprender las relaciones entre diferentes elementos. Además, resultan muy útiles para resolver problemas relacionados con la búsqueda de caminos más cortos, lo cual facilita la planificación de rutas óptimas. La eficiencia en el procesamiento de grafos depende en gran medida del algoritmo utilizado. Entre los algoritmos más eficientes, se destaca el algoritmo de Dijkstra, el cual optimiza diversos procesos y permite encontrar la mejor solución a un problema dado.
Para mejorar la eficiencia en el manejo de grafos, es posible aplicar técnicas de optimización de la complejidad computacional, como el uso de estructuras de datos auxiliares, por ejemplo, colas de prioridad, las cuales permiten un acceso rápido al elemento de mayor prioridad y mantienen un orden específico.

Tablas Hash,

Las tablas hash constituyen estructuras de datos que nos brindan un acceso eficiente a los elementos almacenados, gracias a la asignación de claves únicas mediante una función hash. En promedio, presentan una complejidad computacional constante (O(1)), lo cual las convierte en estructuras muy eficientes. Sin embargo, en el peor de los casos, su complejidad puede llegar a ser lineal (O(n)), dependiendo de la cantidad de colisiones presentes en el conjunto de datos. Para mejorar la eficiencia de las tablas hash, es posible emplear enfoques más sofisticados, como el uso de tablas hash más avanzadas que distribuyan los elementos de manera más uniforme, o incluso la implementación de algoritmos específicos para manejar las colisiones, como el uso de árboles binarios de búsqueda.

Como mejorarlas

Linked list,

Implementar una linkedlist con operaciones optimizadas para inserción y eliminación en el inicio y final de la lista.
Utilizar una linkedlist circular en lugar de una linkedlist simple para mejorar el acceso a ambos extremos de la lista.
Implementar una técnica llamada "skip list" que permite un acceso más rápido a elementos específicos mediante el uso de múltiples capas de nodos.

Doubly linked list,

Implementar una variante llamada "circular doubly linked list" que conecte el último nodo con el primero y el primer nodo con el último, lo que permite un acceso más eficiente a ambos extremos de la lista.Utilizar una técnica llamada "lazy deletion" para marcar los nodos eliminados en lugar de eliminarlos físicamente, lo que evita la necesidad de reorganizar las referencias de los nodos adyacentes.

Árboles de búsqueda binaria,

Implementar variantes de árboles balanceados, como los árboles AVL, árboles rojo-negro o árboles B, que garantizan un balance adecuado y minimizan la altura del árbol.
Utilizar técnicas de optimización de búsqueda, como el árbol de búsqueda binario óptimo, que minimiza la cantidad esperada de comparaciones en la búsqueda de elementos.
Implementar una variante llamada "árbol de búsqueda binaria autoajustable" que realiza rotaciones y reestructuraciones automáticamente para mantener el árbol balanceado durante las operaciones de inserción y eliminación.

Grafos,

Utilizar algoritmos de búsqueda más eficientes, como el algoritmo de búsqueda en anchura (BFS) o el algoritmo de búsqueda en profundidad (DFS), para encontrar caminos o recorrer el grafo de manera óptima.
Aplicar técnicas de poda o reducción del grafo para eliminar nodos o aristas que no son relevantes para el problema, lo que reduce el tamaño y la complejidad del grafo.
Implementar estructuras de datos auxiliares, como matrices de adyacencia o listas de adyacencia optimizadas, para representar y manipular grafos de manera más eficiente.

Tablas Hash,

Utilizar funciones de dispersión más sofisticadas y de alta calidad para minimizar las colisiones y distribuir los elementos de manera más uniforme en la tabla.
Implementar técnicas de resolución de colisiones como "encadenamiento" o "resolución abierta" con sondas lineales o dobles, que permiten manejar las colisiones de manera más eficiente.
Utilizar técnicas de re hashing dinámico para redimensionar la tabla y mantener una carga óptima, evitando así colisiones y garantizando un rendimiento constante.

Conclusión,

En este reporte final se reflexiona sobre el curso de estructuras de datos, destacando la importancia de comprender las características y aplicaciones de diferentes estructuras. Se resalta que no existe una estructura de datos perfecta y que cada una tiene fortalezas y debilidades dependiendo del escenario en el que se utilice.

En cuanto a los algoritmos de ordenamiento, se mencionan varios, como bubble sort, merge sort, quick sort e insertion sort, y se destaca la importancia de elegir el algoritmo adecuado considerando la complejidad temporal, el tamaño del conjunto de datos y su estructura. Se menciona que quicksort y mergesort son algoritmos eficientes con una complejidad promedio de O(N log N), mientras que bubble sort e insertion sort tienen una complejidad cuadrática.

En relación a las estructuras de datos lineales, se habla de las listas enlazadas y los vectores, resaltando las ventajas y desventajas de cada una. Se menciona la posibilidad de mejorar la eficiencia de las listas enlazadas mediante el uso de vectores dinámicos.

En cuanto a los árboles de búsqueda binaria, se destaca su eficiencia en operaciones de búsqueda y se menciona la importancia del balanceo para mejorar su rendimiento. Se menciona la existencia de árboles AVL, que mantienen un equilibrio de altura.
En relación a los grafos, se resalta su versatilidad y utilidad para representar relaciones complejas. Se menciona el algoritmo de Dijkstra como un ejemplo de algoritmo eficiente para encontrar caminos más cortos.

En cuanto a las tablas hash, se menciona su eficiencia en el acceso a los elementos, pero se advierte sobre la posibilidad de colisiones. Se menciona la utilización de funciones de dispersión más sofisticadas y técnicas de resolución de colisiones para mejorar la eficiencia de las tablas hash.

En general, se destaca la importancia de elegir sabiamente la estructura de datos y considerar tanto la complejidad computacional como el costo computacional. Además, se mencionan diversas técnicas para mejorar la eficiencia de las estructuras y algoritmos, como la selección adecuada del pivote, el uso de estructuras auxiliares y la implementación de variantes más avanzadas.

Referencias apa,

- *Estructura de Datos : Grafos - Fundamentos*. (s. f.). https://www.fceia.unr.edu.ar/estruc/2005/graffund.htm
- *5.5. Transformación de claves (hashing) — Solución de problemas con algoritmos y estructuras de datos*. (s. f.). https://runestone.academy/ns/books/published/pythoned/SortSearch/TransformacionDeClaves.html
- *Árboles Binarios*. (2014, 10 junio). Estructuras de datos. https://hhmosquera.wordpress.com/arbolesbinarios/
- Medina, R. (2020). Estructura de Datos - Linked List (Lista Enlazada). *DEV Community*. https://dev.to/ronnymedina/estructura-de-datos-linked-list-lista-enlazada-2h9
- GeeksforGeeks. (2023). Introduction to Doubly Linked List Data Structure and Algorithm Tutorials. *GeeksforGeeks*. https://www.geeksforgeeks.org/data-structures/linked-list/doubly-linked-list/
- *LUDA UAM-Azc*. (s. f.). http://aniei.org.mx/paginas/uam/CursoPoo/curso_poo_12.html#:~:text=Un%20%C3%A1rbol%20binario%20es%20un,acceder%20a%20sus%20elementos%20ordenadamente.
- *DSTool: Herramienta para la programación con estructuras de datos*. (s. f.). http://www.hci.uniovi.es/Products/DSTool/hash/hash-queSon.html
- *TABLAS HASH*. (s. f.). https://ccia.ugr.es/~jfv/ed1/tedi/cdrom/docs/tablash.html#:~:text=El%20rehashing%20consiste%20en%20que,%2Crehi(k).