Tecnológico de Monterrey

Act 1.3 - Actividad Integral de Conceptos Básicos y Algoritmos Fundamentales (Evidencia Competencia)

Aaron Hernandez Jimenez

A01642529

31 de agosto del 2023

Programación de estructuras de datos y algoritmos fundamentales (Gpo 602)

Jorge Enrique González Zapata

Cosas importantes a tener en cuenta antes de ejecutar el código:

hay 3 casos que pueden ocurrir, el primero es en el cual no se encuentra un archivo de orden por lo cual no se ha ordenado el arreglo anteriormente, el segundo es cuando se encuentra el archivo nuevo pero este se encuentra desordenado para estos 2 casos el programa va ejecutar el algoritmo Bubblesort el cual dura aproximadamente 2:30 min, por lo cual se recomienda iniciar el ejecutable con los archivos incluidos ya que estos son clave para que no vuelva a iniciar el programa de organización, el tercer caso es en el cual se encuentra el archivo ordenado y está efectivamente ordenado, ahí pasará a preguntar al usuario si quiere una fecha preestablecida o una ingresada por el, en el caso de ingresar 1 buscará una fecha predeterminada, en caso de ingresar 0 pedirá el ingreso de la fecha inicial y la fecha final en formato de 2 dígitos, por ejemplo:
mes 1
dia 12
hora 20
minuto 10.

El programa genera un archivo de texto aparte en el cual se ingresarán los registros entre las fechas definidas por el usuario, el archivo tiene nombre de Búsqueda Por Fecha.txt y al inicio del mismo indica entre cuales fechas se buscó.

Documento

He realizado una investigación y reflexión sobre la importancia y eficiencia del algoritmo de ordenamiento Quicksort en situaciones problemáticas.

La importancia de Quicksort:

El algoritmo de ordenamiento Quicksort es uno de los algoritmos más importantes y eficientes en el campo de la informática y la ciencia de la computación. Su importancia radica en varios aspectos clave:

- **Eficiencia Promedio:** Quicksort es conocido por su eficiencia en promedio. Tiene un rendimiento excelente en una amplia gama de situaciones, con una complejidad media de tiempo de O(n log n). Esto significa que puede ordenar grandes conjuntos de datos de manera rápida y eficiente.

- **Divide and Conquer:** Quicksort utiliza la estrategia de "dividir y vencer". Divide el conjunto de datos en subconjuntos más pequeños, los ordena de manera independiente y luego combina los resultados. Esta estrategia lo hace altamente paralelizable y adecuado para sistemas multiproceso.

- **Adaptabilidad:** Quicksort puede ser altamente adaptable a diferentes situaciones y tipos de datos. Puede optimizarse para mejorar su rendimiento en datos casi ordenados o parcialmente ordenados.

- ➔ **Espacio en Memoria:** Quicksort es un algoritmo in situ, lo que significa que no requiere memoria adicional para realizar la clasificación. Esto lo hace eficiente en términos de uso de memoria.

Eficiencia en la búsqueda:

Aunque Quicksort es principalmente un algoritmo de ordenamiento, su estructura también se utiliza en la búsqueda. El algoritmo Quicksort puede ser modificado para realizar búsqueda binaria en una lista ordenada. Esto significa que podemos buscar elementos en una lista ordenada de manera eficiente, reduciendo la complejidad de tiempo de O(n) de una búsqueda lineal a O(log n).

Reflexión Personal:

Considero que el estudio y comprensión de algoritmos de ordenamiento como Quicksort son fundamentales. No solo me permite desarrollar habilidades para resolver problemas de manera eficiente, sino que también me proporciona una comprensión más profunda de cómo funcionan los algoritmos y cómo pueden adaptarse a situaciones específicas.

El Quicksort destaca como un algoritmo versátil y eficiente, pero también es importante recordar que no es el único algoritmo de ordenamiento. En la práctica, la elección del algoritmo adecuado depende del tipo de datos, el tamaño del conjunto de datos y los requisitos específicos del problema.

Situación Problema: Supongamos que estás desarrollando una aplicación de comercio electrónico y necesitas ordenar rápidamente una lista de productos en función de su precio. Los productos se actualizan con frecuencia, por lo que necesitas un algoritmo de ordenamiento eficiente para que los usuarios puedan ver los productos en orden de precio en tiempo real.

Uso de QuickSort: En esta situación, QuickSort sería una elección sólida debido a su eficiencia en el rendimiento promedio. Puede ordenar rápidamente la lista de productos en función de los precios, lo que permite a los usuarios ver los productos más caros o más baratos en segundos, incluso cuando la lista es grande.

La importancia de Mergesort:

Su importancia se basa en varios aspectos clave:

- **Estabilidad y Predecibilidad:** Mergesort es un algoritmo estable, lo que significa que no cambia el orden relativo de elementos iguales. Esta propiedad es crucial en aplicaciones que requieren mantener relaciones de orden específicas, como bases de datos y sistemas que manejan datos sensibles.
- **Eficiencia Asintótica:** Mergesort demuestra su valor en términos de tiempo y espacio. Tiene un rendimiento consistente con una complejidad de tiempo de O(n log n) en el peor caso, lo que lo hace adecuado para conjuntos de datos de mayor tamaño.

- **Divide and conquer:** Mergesort se adhiere al enfoque "divide y vencerás", descomponiendo el problema en subproblemas más pequeños y luego fusionando los resultados. Esta estrategia no solo facilita su implementación, sino que también permite la paralelización en sistemas multiproceso, esto ya lo habíamos visto aplicado en Quicksort y es parecida la estrategia que se aplica.
- **Adaptabilidad:** Mergesort es adecuado para una variedad de situaciones. Aunque puede requerir más memoria en comparación con otros algoritmos, su rendimiento constante lo hace atractivo en escenarios donde se prioriza la eficiencia sobre el uso de memoria.

Eficiencia en la búsqueda:

Aunque Mergesort es conocido principalmente como un algoritmo de ordenamiento, su estructura también se ha utilizado en la búsqueda. La versión modificada de Mergesort puede llevar a cabo una búsqueda eficiente en una lista ordenada, logrando una complejidad de tiempo de O(log n) en la búsqueda.

Reflexión de Mergesort:

La eficiencia de Mergesort es innegable en términos de ordenamiento y búsqueda en listas ordenadas. Su estabilidad y predictibilidad son cualidades que pueden ser cruciales en sistemas que requieren consistencia y control. Sin embargo, también reconozco que la elección del algoritmo de ordenamiento adecuado depende de varios factores, como el tipo de datos y los requerimientos del problema.

Situación Problema: Imagina que trabajas en una biblioteca digital que almacena y ordena una gran cantidad de libros electrónicos en función de sus títulos. Los usuarios desean buscar libros por título, y necesitas garantizar que la lista esté siempre ordenada alfabéticamente para facilitar las búsquedas.

Uso de MergeSort: MergeSort sería una elección adecuada en esta situación debido a su estabilidad y eficiencia en la ordenación. Puedes utilizar MergeSort para mantener la lista de libros electrónicos ordenada por título de manera constante, lo que facilita la búsqueda y recuperación eficiente de libros para los usuarios.

La importancia de Timsort:

- ➔ **Eficiencia y Adaptabilidad:** Timsort es un algoritmo de ordenamiento híbrido que combina las fortalezas de MergeSort y InsertionSort. Esto lo hace eficiente en una amplia gama de situaciones. Su complejidad de tiempo en el peor caso es O(n log n), lo que garantiza un buen rendimiento en conjuntos de datos de diferentes tamaños.

- ➔ **Estabilidad:** Al igual que MergeSort, Timsort es un algoritmo estable. Mantiene el orden relativo de elementos iguales en la lista, lo que es esencial en aplicaciones donde la integridad de los datos es crítica.

- ➔ **Adaptabilidad a Datos Parcialmente Ordenados:** Timsort está diseñado para manejar conjuntos de datos parcialmente ordenados de manera eficiente. Esto lo hace adecuado para aplicaciones del mundo real donde los datos pueden estar desordenados en ciertos momentos.

- ➔ **Optimización para Datos Pequeños:** Timsort incluye una optimización que utiliza InsertionSort para conjuntos de datos pequeños, lo que reduce el consumo de recursos y mejora la eficiencia en estos casos.

- ➔ **El mejor Best Case:** ya que si se encuentra ordenado se vuelve de complejidad de Ω(n) lo cual hace que tenga un rendimiento en arreglos ya pre ordenados.

Eficiencia en la Búsqueda:

Si bien Timsort es principalmente un algoritmo de ordenamiento, su estructura se puede adaptar para mejorar la eficiencia en la búsqueda. La capacidad de mantener el orden relativo de los elementos es beneficiosa al buscar elementos en una lista ordenada.

Reflexión Personal:

La combinación de MergeSort y InsertionSort en Timsort lo convierte en un algoritmo versátil y eficiente que puede manejar conjuntos de datos de diferentes tamaños y niveles de desorden. Su capacidad para mantener la estabilidad en los datos es especialmente valiosa en aplicaciones donde la integridad de la información es esencial.

Si bien Timsort es una herramienta poderosa, también reconozco que cada algoritmo tiene su lugar. La elección del algoritmo de ordenamiento adecuado depende de factores como el tipo de datos y los requerimientos específicos del problema.

Situación Problema: Supongamos que trabajas en el desarrollo de un sistema de gestión de archivos para una empresa que almacena una gran cantidad de documentos. Los documentos se agregan y actualizan constantemente, y necesitas una manera eficiente de organizarlos para que los usuarios puedan encontrar rápidamente lo que necesitan.

Uso de Timsort: En esta situación, Timsort sería una excelente elección debido a su capacidad para manejar diferentes tamaños de conjuntos de datos y su adaptabilidad a datos parcialmente ordenados. Puedes utilizar Timsort para ordenar la lista de documentos por fecha de creación o modificación, lo que permite a los usuarios acceder fácilmente a los documentos más recientes o a los que han sido modificados recientemente. La estabilidad de Timsort garantiza que los documentos con la misma fecha se mantengan en el orden original, lo que puede ser importante en aplicaciones de gestión de archivos.

Las Limitaciones de Bubble Sort:

- ➔ **Ineficiencia en Conjuntos de Datos Grandes:** Uno de los problemas más notorios de Bubble Sort es su ineficiencia en conjuntos de datos grandes. Su complejidad de tiempo en el peor caso es O(n^2), lo que significa que el tiempo de ejecución aumenta

cuadráticamente con el tamaño del conjunto de datos. En comparación con otros algoritmos más eficientes como MergeSort o Quick Sort, Bubble Sort se queda muy rezagado.

- ➔ **Falta de Adaptabilidad:** Bubble Sort no se adapta bien a diferentes situaciones o tipos de datos. Su rendimiento es pobre tanto en datos ordenados como en datos desordenados, lo que lo convierte en una opción poco adecuada para aplicaciones del mundo real donde los datos pueden variar.
- ➔ **Estabilidad en Rendimiento:** Bubble Sort tiene un rendimiento constante y predecible, pero en el sentido negativo. Siempre realizará una cantidad fija de comparaciones y swaps, independientemente de si los datos ya están ordenados o no. Esto significa que incluso en casos donde otros algoritmos más eficientes podrían ahorrar tiempo, Bubble Sort seguirá ejecutando el mismo número de operaciones.
- ➔ **Uso de Espacio:** Aunque Bubble Sort es in situ (no requiere memoria adicional), su ineficiencia en tiempo significa que los programas que lo utilizan pueden ejecutarse durante más tiempo, lo que puede ser un costo oculto en términos de recursos de la CPU.

Reflexión Personal:

He llegado a la conclusión de que Bubble Sort es uno de los algoritmos de ordenamiento menos eficientes y menos versátiles disponibles. A pesar de su simplicidad conceptual, su falta de adaptabilidad y su ineficiencia en conjuntos de datos más grandes lo hacen inadecuado para la mayoría de las aplicaciones del mundo real.

Si bien es importante entender y aprender de Bubble Sort como estudiantes, también es esencial reconocer sus limitaciones y optar por algoritmos más eficientes y efectivos en situaciones prácticas.

Aunque es didáctico en la enseñanza de algoritmos, carece de la eficiencia y la adaptabilidad necesarias en aplicaciones del mundo real. Como estudiante, reconozco la importancia de comprender sus limitaciones y optar por algoritmos más adecuados para problemas complejos y conjuntos de datos grandes.

Situación Problema: Supongamos que estás creando una aplicación educativa para enseñar a los niños a ordenar una serie de números. La aplicación debe mostrar el proceso de ordenamiento paso a paso de manera visual y didáctica.

Uso de BubbleSort: Aunque BubbleSort no es eficiente en términos de rendimiento para conjuntos de datos grandes, puede ser útil en situaciones donde se necesita un algoritmo simple para enseñar conceptos de ordenamiento. Puedes usar BubbleSort en esta aplicación educativa para demostrar visualmente cómo se ordenan los números y cómo funciona un algoritmo de ordenamiento.

Las Fortalezas de CubeSort:

- ➔ **Complejidad de Tiempo Razonable:** CubeSort ofrece una complejidad de tiempo de O(n log n), lo que lo sitúa en la misma categoría de eficiencia que algoritmos bien

conocidos como MergeSort y QuickSort. Esto significa que, en el peor de los casos, el tiempo de ejecución aumenta de manera logarítmica con el tamaño del conjunto de datos, lo que lo hace adecuado para manejar conjuntos de datos de diferentes tamaños.

- ➔ **Estabilidad en el rendimiento:** A diferencia de BubbleSort, CubeSort muestra una adaptabilidad razonable en diferentes situaciones. Aunque no es tan versátil como QuickSort o MergeSort, CubeSort mantiene un rendimiento constante y predecible en una variedad de conjuntos de datos, lo que lo convierte en una opción sólida para aplicaciones que requieren estabilidad en el rendimiento.
- ➔ **Mantenimiento de la Memoria:** CubeSort es un algoritmo in situ, lo que significa que no requiere memoria adicional para realizar la ordenación. Esto lo hace eficiente en términos de uso de recursos de memoria y adecuado para aplicaciones con limitaciones de memoria.
- ➔ **Adaptabilidad a Datos Preordenados:** A diferencia de algunos algoritmos de ordenamiento que pueden ralentizar significativamente cuando se les da una lista que ya está preordenada, CubeSort mantiene un rendimiento razonable en esta situación. Esto lo hace útil en aplicaciones donde los datos a menudo están parcialmente ordenados.

Reflexión Personal:

Basado en mi investigación y análisis, considero que CubeSort es un algoritmo de ordenamiento sólido y eficiente que tiene su lugar en el mundo de los algoritmos. Aunque no es tan versátil como algunos otros algoritmos, ofrece un equilibrio razonable entre eficiencia y estabilidad en el rendimiento.

CubeSort, aunque quizás no sea tan conocido como algunos otros algoritmos, es una herramienta valiosa en la caja de herramientas de un estudiante y un profesional de la informática. Su equilibrio entre eficiencia y adaptabilidad lo hace relevante en una variedad de situaciones problema y es digno de estudio y consideración en aplicaciones del mundo real.

Situación Problema: Supongamos que trabajas en un sistema de gestión de inventario de una tienda en línea que maneja una gran cantidad de productos. Quieres ordenar los productos en función de sus categorías y precios, y necesitas una solución que sea eficiente incluso cuando los datos no están completamente ordenados.

Uso de CubeSort: CubeSort podría ser una elección adecuada en esta situación debido a su eficiencia en el manejo de datos parcialmente ordenados. Puedes utilizar CubeSort para ordenar los productos primero por categoría y luego por precio dentro de cada categoría, lo que facilita la navegación y búsqueda de productos para los clientes en la tienda en línea.