Implementación del Algoritmo de Mezcla Directa para Ordenar Archivos Externos

Introducción:

El procesamiento de grandes conjuntos de datos es una tarea común en la informática y la ciencia de datos. Cuando se trata de ordenar grandes cantidades de información almacenadas en archivos, es necesario utilizar algoritmos de ordenamiento externo que puedan manejar la limitación de la memoria principal. Uno de estos algoritmos es la "Mezcla Directa". A continuación se presenta una implementación de este algoritmo para ordenar un archivo de números enteros almacenados de manera desordenada. Se explicará la solución utilizada y se evaluarán los resultados obtenidos.

Solución:

1. *Generación de Datos Desordenado:* Se inicia generando un archivo de datos desordenado que contiene números enteros aleatorios. Estos números se generan en base a los parámetros proporcionados, incluyendo la cantidad de líneas y la cantidad de cifras de cada número. Los números se escriben en el archivo, uno por línea, tal que el archivo tenga un tamaño mayor o igual a 100 megabytes.
2. *Algoritmo de Mezcla Directa:* El algoritmo de Mezcla Directa se utiliza para ordenar el archivo de datos desordenados. El algoritmo opera de la siguiente manera:
    - Lee el archivo en bloques de tamaño B (un parámetro definido) y ordena cada bloque internamente.
    - Los bloques ordenados se almacenan en una lista.
    - Se realiza una fusión iterativa de los bloques ordenados, asegurándose de que la primera línea del archivo resultante contenga el número entero más pequeño y la última línea el número más grande.

3. *Verificación del Resultado:* Una vez que se ha ordenado el archivo, se realiza una verificación para asegurarse de que el proceso se haya completado con éxito. Esto implica comparar el tamaño del archivo ordenado con el tamaño del archivo original (ambos pasados a bytes) y verificar que los datos estén correctamente ordenados de menor a mayor.

Resultados:

La implementación del algoritmo de Mezcla Directa se ha ejecutado con éxito en un archivo de 5 millones de números enteros generados aleatoriamente. El proceso de ordenamiento se llevó a cabo eficientemente, y el archivo resultante contiene los números enteros ordenados de menor a mayor.

Conclusiones:

Lo más valorable es que el código cumple su propósito principal de generar, ordenar y verificar archivos con números enteros, pero identificamos áreas que pueden mejorarse. En particular, la gestión de la memoria podría ser más eficiente, especialmente para archivos grandes. La implementación de pruebas unitarias es fundamental para garantizar la funcionalidad y detectar posibles errores. Además, si se considera un uso más interactivo, agregaríamos una interfaz de usuario.