Resumen de la semana 12

Tema: Ordenación y Ordenación por Fusión

7.4. Ordenación

Los algoritmos de ordenación, que organizan los elementos de una lista o arreglo en un orden específico, típicamente ascendente o descendente.

La ordenación es fundamental en informática, ya que optimiza operaciones como la búsqueda, la fusión de datos y el procesamiento eficiente. Los algoritmos se evalúan según su complejidad temporal (tiempo de ejecución), complejidad espacial (uso de memoria) y estabilidad, que indica si se preserva el orden relativo de elementos con valores iguales.

Existen dos categorías principales de algoritmos: los basados en comparaciones (como Bubble Sort, Insertion Sort, Merge Sort y Quicksort) y los no comparativos (como Counting Sort o Radix Sort).

La elección del algoritmo depende del tamaño del arreglo, la distribución de los datos, los requisitos de memoria y la necesidad de estabilidad. Por ejemplo, Insertion Sort es eficiente para arreglos pequeños, mientras que Merge Sort y Quicksort son preferidos para conjuntos de datos grandes.

7.4.1. Ordenación por Fusión

La ordenación por fusión (*Merge Sort*) es un algoritmo basado en la técnica de

divide y vencerás, estructurado en tres etapas:

División: El arreglo se divide recursivamente en dos mitades hasta obtener

subarreglos de un solo elemento o vacíos.

Resolución: Los subarreglos de un elemento están ordenados por definición.

Fusión: Los subarreglos se combinan en un arreglo ordenado, comparando

elementos y colocándolos en la posición correcta.

La fusión, el paso clave, compara los elementos más pequeños de cada subarreglo,

seleccionándolos para el arreglo resultante, y avanza en el subarreglo correspondiente.

Este proceso requiere un arreglo auxiliar, lo que incrementa el uso de memoria.

Merge Sort tiene una complejidad temporal de $O(n \log n)$ en todos los casos (mejor, promedio y peor), ya que la división genera $\log n$ niveles y la fusión en cada nivel cuesta $O(n)$. Esto lo hace más predecible que *Quicksort*, que puede degradarse a $O(n2)$.

Sin embargo, *Merge Sort* no es *in-place*, requiriendo $O(n)$ de memoria auxiliar, una desventaja en sistemas con recursos limitados.

Su estabilidad, que preserva el orden relativo de elementos con valores iguales, lo hace ideal para aplicaciones como la ordenación de registros por múltiples claves.

Es eficiente para listas enlazadas, ya que no requiere acceso aleatorio, y para datos en almacenamiento externo, donde el acceso secuencial es óptimo.

Merge Sort se utiliza en bases de datos, sistemas distribuidos y algoritmos híbridos, especialmente cuando la estabilidad o el procesamiento de grandes volúmenes de datos es crucial. Su diseño lo hace robusto, aunque su costo en memoria debe considerarse en entornos con restricciones.

Ejemplo Práctico Resuelto

Ejemplo: Ordenar el arreglo [8, 3, 5, 1, 9, 2, 7, 4] usando Merge Sort.

Solución: A continuación, se detallan los pasos, con cada acción en una línea separada para mayor claridad:

1. División inicial: Dividimos el arreglo [8, 3, 5, 1, 9, 2, 7, 4] en dos mitades.

2. Subarreglo izquierdo: Obtenemos [8, 3, 5, 1].

3. Subarreglo derecho: Obtenemos [9, 2, 7, 4].

4. División del subarreglo izquierdo: Dividimos [8, 3, 5, 1] en [8, 3] y [5, 1].

5. División de [8, 3]: Obtenemos [8] y [3].

6. División de [5, 1]: Obtenemos [5] y [1].

7. División del subarreglo derecho: Dividimos [9, 2, 7, 4] en [9, 2] y [7, 4].

8. División de [9, 2]: Obtenemos [9] y [2].

9. División de [7, 4]: Obtenemos [7] y [4].

10. Fusión de [8] y [3]: Comparamos 8 > 3, resultando en [3, 8].

11. Fusión de [5] y [1]: Comparamos 5 > 1, resultando en [1, 5].

12. Fusión de [3, 8] y [1, 5]: Comparamos 3 > 1, 3 < 5, 8 > 5, resultando en [1, 3, 5, 8].

13. Fusión de [9] y [2]: Comparamos 9 > 2, resultando en [2, 9].

14. Fusión de [7] y [4]: Comparamos 7 > 4, resultando en [4, 7].

15. Fusión de [2, 9] y [4, 7]: Comparamos 2 < 4, 4 < 9, 7 < 9, resultando en [2, 4, 7, 9].

16. Fusión final de [1, 3, 5, 8] y [2, 4, 7, 9]: Comparamos 1 < 2, 3 > 2, 3 < 4, 5 > 4, 5 < 7, 8 > 7, 8 < 9, resultando en [1, 2, 3, 4, 5, 7, 8, 9].

17. Resultado final: El arreglo ordenado es [1, 2, 3, 4, 5, 7, 8, 9].

Este ejemplo muestra cómo Merge Sort divide el arreglo en partes manejables y las recombina ordenadamente, garantizando estabilidad y un rendimiento constante.

Conclusión

La ordenación es un componente esencial en algoritmia, y *Merge Sort* destaca por su complejidad $O(n \log n)$, estabilidad y adaptabilidad a estructuras como listas enlazadas o datos externos.

Aunque requiere memoria auxiliar, su predictibilidad lo hace ideal para aplicaciones críticas, como bases de datos o sistemas distribuidos. El ejemplo práctico, con pasos claramente separados, ilustra su proceso sistemático, facilitando la comprensión de su mecánica.

Comparado con otros algoritmos, *Merge Sort* ofrece un equilibrio entre estabilidad y eficiencia, siendo una elección robusta para escenarios donde la consistencia es clave.