Ordenamiento

ESTRUCTURA DE DATOS

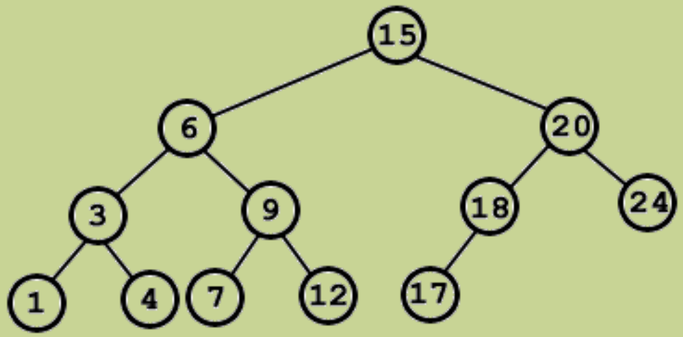
Métodos de ordenamiento

- ShellSort
- QuickSort
- HeapSort
- Radix

ShellSort

El método ShellSort es un algoritmo de ordenamiento que mejora el método de inserción al permitir intercambios a mayores distancias entre elementos. Dividir la lista en sublistas más pequeñas y ordenarlas, luego realizar una pasada final con sublista de tamaño 1. Es más eficiente que el método de inserción estándar pero no tan rápido como otros algoritmos avanzados.

QuickSort



Es un algoritmo de ordenamiento muy eficiente que utiliza el enfoque "divide y vencerás". Funciona seleccionando un elemento como pivote, particionando la lista alrededor del pivote y luego aplicando recursivamente este proceso a las sublistas resultantes. Tiene una complejidad promedio de tiempo de O(n log n), lo que lo hace rápido en la mayoría de los casos. Es ampliamente utilizado debido a su eficiencia, especialmente en conjuntos de datos grandes.

HeapSort

Es un algoritmo de ordenamiento que utiliza una estructura de datos llamada montículo (heap). Funciona extrayendo repetidamente el máximo (en un montículo máximo) y reorganizando los elementos restantes. Tiene una complejidad de tiempo de O(n log n), haciéndolo eficiente para grandes conjuntos de datos, pero no es tan popular como otros algoritmos como QuickSort debido a su complejidad de implementación.

Radix

Es un algoritmo de ordenamiento no comparativo que ordena los elementos procesando los dígitos individuales de los números. Clasifica los elementos en baldes según el valor de sus dígitos en diferentes posiciones y luego combina los baldes en orden. Es eficiente para ordenar números enteros no negativos y puede tener un rendimiento lineal en el mejor de los casos. Sin embargo, no es tan comúnmente utilizado como otros algoritmos de ordenamiento.

TABLA COMPARATIVA

Método	Ventajas	Desventajas	Descripción	Big O
HellSort	• Eficiente en listas medianas • Es más fácil de implementar que algoritmos más avanzados como Quicksort o Heapsort.	• No tan eficiente en listas pequeñas y grandes • No es estable, lo que significa que puede cambiar el orden relativo de elementos con claves iguales.	Divide la lista en sublistas y aplica ordenamiento por inserción	$O(n\ Log(n))$
QuickSort	• Quicksort es flexible en términos de selección de pivotes. • Quicksort es un algoritmo in situ, lo que significa que no requiere memoria adicional para ordenar la lista.	• En el peor caso, Quicksort puede degradarse a un tiempo de ejecución cuadrático de O(n^2), • Quicksort no conserva el orden relativo de elementos con claves iguales.	Elección del pivote: Selecciona un elemento de la lista como pivote. Partición: Reorganiza la lista de manera que todos los elementos más pequeños que el pivote estén a su izquierda y todos los elementos más grandes estén a su derecha.	
HeapSort	• Sirve para conjuntar datos grandes • Eficiencia en el tiempo de ejecución • No necesita más memoria	• No es óptimo para listas pequeñas debido a su sub inicial • Complejidad de implementación • No es adaptivo	La matriz lo convierte es un heap independiendo si es ordenada ente se base en heap o mínimo. Luego obtiene el elemento y lo pone al final se reapare, ya cuando se extrajeron todos los elementos heap, se ordena.	
Radix	• Su tiempo de ejecución es lineal • Sirve para numerar grandes con muchos dígitos haciendo menos operaciones que otros métodos.	• El tamaño no es variable • Requiere más espacio de memoria • No es eficiente con números pequeños	Clasifica los elementos tomando en cuenta sus dígitos individuales	$B(n+k)$

Aplicaciones

QuickSort

Biblioteca de una librería: Imagina que tienes una gran cantidad de libros en una biblioteca y deseas organizarlos por orden alfabético según el título. Puedes aplicar QuickSort para ordenar los libros rápidamente. Seleccionas un libro como "pivote", colocas los libros cuyos títulos están antes del pivote en una pila y los que están después en otra, y luego repites el proceso para cada pila hasta que todos los libros estén ordenados.

Aplicaciones

ShellSort

Ordenando documentos en una oficina: Supongamos que en una oficina tienes muchos documentos que necesitas ordenar por fecha. ShellSort podría ser útil aquí. Divide los documentos en grupos más pequeños y ordena cada grupo individualmente utilizando algún criterio, como el mes. Luego, combina los grupos y continúa ordenando hasta que todos los documentos estén en orden cronológico.

Aplicaciones

Radix

Organizando archivos digitales por tamaño: Si tienes una gran cantidad de archivos digitales en una computadora y deseas organizarlos por tamaño, Radix Sort podría ser útil. En lugar de considerar el tamaño completo de cada archivo, podrías dividirlo en "dígitos" representados por bytes o kilobytes. Luego, ordenarías los archivos en grupos según sus "dígitos" de tamaño y los combinarías en orden para obtener una lista ordenada de archivos según su tamaño.

Muchas gracias

LUIS SALAZAR