Tema investigación: Métodos de ordenación y búsqueda (Heap sort).

Estudiante: Juan Esteban Valverde Mora.

Carrera: Ingeniería Informática.

Ciclo lectivo: 1.er cuatrimestre 2025, Estructuras de Datos.

Profesor: Juan Diego Pacheco Jiménez.

Fecha de entrega: 31/03/2025.

Índice

Introducción

En la informática, una de las cosas fundamentales en el manejo de grandes cantidades de datos es la ordenación. Los algoritmos de ordenación permiten organizar los datos de manera eficiente, haciendo fácil su acceso, búsqueda y manipulación, en resumen, toda su gestión. En la práctica, los algoritmos de ordenación se usan en muchos campos, desde bases de datos hasta sistemas de archivos.

Existen diversos algoritmos para ordenar datos, cada uno con características distintas en cuanto a eficiencia y tiempo, además de la complejidad en su aplicación. Entre los algoritmos más conocidos se encuentran BubbleSort, QuickSort, MergeSort, RadixSort, y SelectionSort, etc. Sin embargo, uno de los métodos más interesantes por su eficiencia y su elegancia en la implementación es el algoritmo HeapSort.

HeapSort se basa en una estructura de datos llamada heap, que es un tipo especial de árbol binario. El algoritmo de HeapSort aprovecha esta estructura para ordenar los elementos de un arreglo de manera eficiente, lo que lo convierte en una opción ideal para grandes volúmenes de datos.

Este método tiene ventajas significativas en escenarios donde es necesario un ordenamiento sin necesidad de memoria adicional y donde la eficiencia en tiempo es crucial.

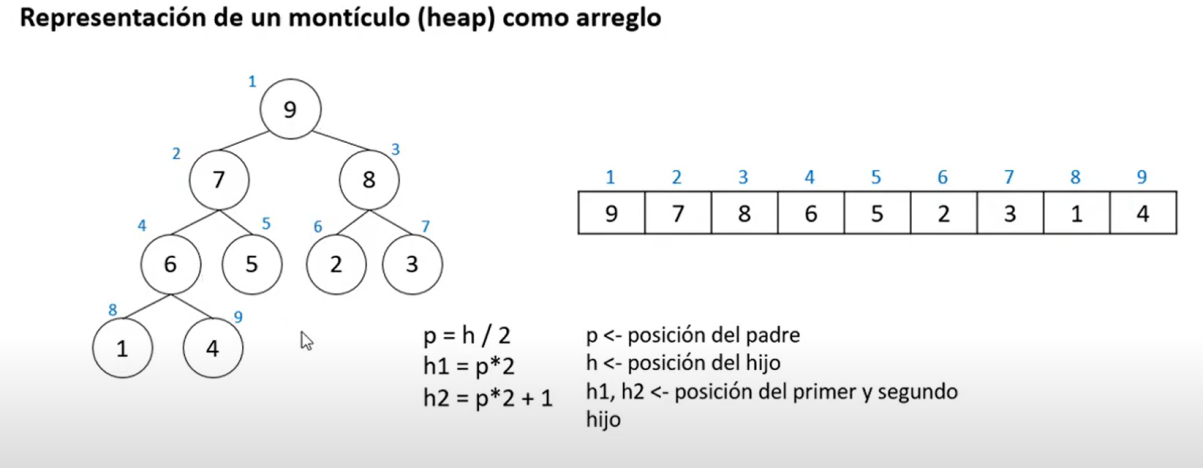
El objetivo de este trabajo es entender su funcionamiento. Para ello, además de este documento con la explicación, se presentará un ejemplo de código en Java que muestre el funcionamiento del método de ordenamiento HeapSort.

Contenido

¿Qué es la recursividad?

La recursividad es un concepto en programación donde una función se llama a sí misma para resolver un problema más pequeño del mismo tipo. Esto es útil cuando un problema puede dividirse en subproblemas más pequeños de forma similar al problema original. Los algoritmos recursivos incluyen una "condición" que termina las llamadas recursivas cuando se logre el objetivo, evitando así un bucle infinito.

Explicación del método Heap sort



En la imagen anterior se puede ver un heap, que tiene la forma como de las raíces de un árbol. Este heap tiene nodos padre y nodos hijo. Cada nodo tiene una posición que se otorga de arriba hacia abajo, de izquierda a derecha. Como se observa ahí, el nodo con el valor 9 tiene la posición 1, sus hijos son los valores 7 y 8 con posiciones 2 y 3, y así sucesivamente con sus hijos.

Esto tiene que cumplir una condición: los nodos hijos deben de tener un valor más pequeño que su padre, en caso contrario se intercambia el valor más alto de sus hijos con el valor del padre. Esta condición se evalúa hasta llegar a la cima, haciendo así que el valor en la cabeza del árbol sea el valor más alto de los nodos. Una vez en la cima se "remueve" el valor en la cabeza, y se coloca en la cabeza la última posición del heap, se repite el proceso nuevamente hasta que no quede ningún valor en el heap. Cada valor extraído se va colocando a la izquierda del anterior, dando como resultado una lista ordenada con los valores de menor a mayor.

Las fórmulas que se ven en la imagen nos ayudan a saber cuál es el valor del nodo padre de uno de sus hijos o, en caso contrario, cuáles son los hijos de algún nodo padre. Para ello se utilizan las posiciones de cada nodo.

Usos más importantes del heap sort

Ordenación in-place sin necesidad de memoria adicional

Una de las principales ventajas de HeapSort es que realiza la ordenación in-place, es decir, no requiere memoria adicional (a diferencia de algoritmos como MergeSort, que necesita un espacio adicional proporcional al tamaño de los datos). Esto lo convierte en una opción adecuada para entornos con restricciones de memoria, donde no se pueden utilizar estructuras de datos adicionales como arreglos auxiliares.

Sistemas en tiempo real y embebidos

En sistemas en tiempo real o embebidos donde el uso de recursos es limitado y no se pueden permitir operaciones costosas de memoria, HeapSort es útil debido a su capacidad para ordenar sin requerir memoria adicional (además de la entrada). Esto hace que el algoritmo sea adecuado para aplicaciones donde los recursos de hardware (memoria y tiempo de procesamiento) son limitados, pero se requiere una ordenación eficiente.

Optimización de procesos de búsqueda

En algunos problemas de búsqueda compleja, como la búsqueda de rangos o de segmentos, HeapSort se puede emplear para ordenar y luego realizar búsquedas eficientes sobre datos ordenados. Aunque no es el algoritmo más común para búsqueda por sí mismo, su eficiencia en la ordenación puede ser aprovechada como parte de un algoritmo de búsqueda más complejo, donde la organización de los datos es crucial.

Ordenación de grandes volúmenes de datos

HeapSort es particularmente útil cuando se necesita ordenar grandes cantidades de datos de manera eficiente. Es ideal para situaciones donde se tienen grandes colecciones de datos y se necesita un rendimiento predecible y eficiente. Aunque otros algoritmos como QuickSort pueden ser más rápidos en algunos casos, HeapSort mantiene su rendimiento incluso con listas grandes, lo que lo convierte en una opción confiable.

Ventajas del Heap sort

Complejidad temporal de O(n log n): A diferencia de algoritmos como BubbleSort y InsertionSort, HeapSort tiene una complejidad logarítmica en su tiempo de ejecución, lo que lo hace mucho más eficiente en la mayoría de los casos.

No requiere espacio adicional: A diferencia de MergeSort, que necesita espacio adicional para almacenar los subarreglos, HeapSort se realiza en el mismo arreglo, utilizando solo espacio constante adicional.

Ordenación in-place: Esto significa que no se requiere memoria adicional para realizar el ordenamiento.

Ejemplo programado

Repositorio del codigo del proyecto

Video demostrativo del proyecto en youtube

Conclusiones

En conclusión, el algoritmo HeapSort es una de las soluciones más eficientes para el problema de ordenación. Aunque HeapSort no es tan popular en la práctica como otros algoritmos como QuickSort debido a que en ciertos casos QuickSort puede ser más rápido, HeapSort tiene varias ventajas que lo hacen indispensable en situaciones específicas.

Una de las principales ventajas de HeapSort es su capacidad para ordenar de manera in-place, es decir, sin necesidad de memoria adicional. Esto lo hace adecuado para sistemas con limitaciones de memoria o en situaciones donde se requieren algoritmos que no utilicen espacio extra para realizar la ordenación, lo cual es especialmente importante cuando se manejan grandes volúmenes de datos.

En resumen, a pesar de que HeapSort no sea el algoritmo más utilizado en el día a día, su elegancia, eficiencia y capacidad para ordenar en memoria lo convierten en una herramienta muy poderosa, especialmente en situaciones donde el rendimiento y la utilización eficiente de los recursos son una prioridad.

Bibliografía

Claudia Nallely Sánchez Gómez. (2020, 18 marzo). *Ordenamiento por montículos (heapsort) con representación de arreglo* [Vídeo]. YouTube. https://www.youtube.com/watch?v=x4J5Mcyzdxk