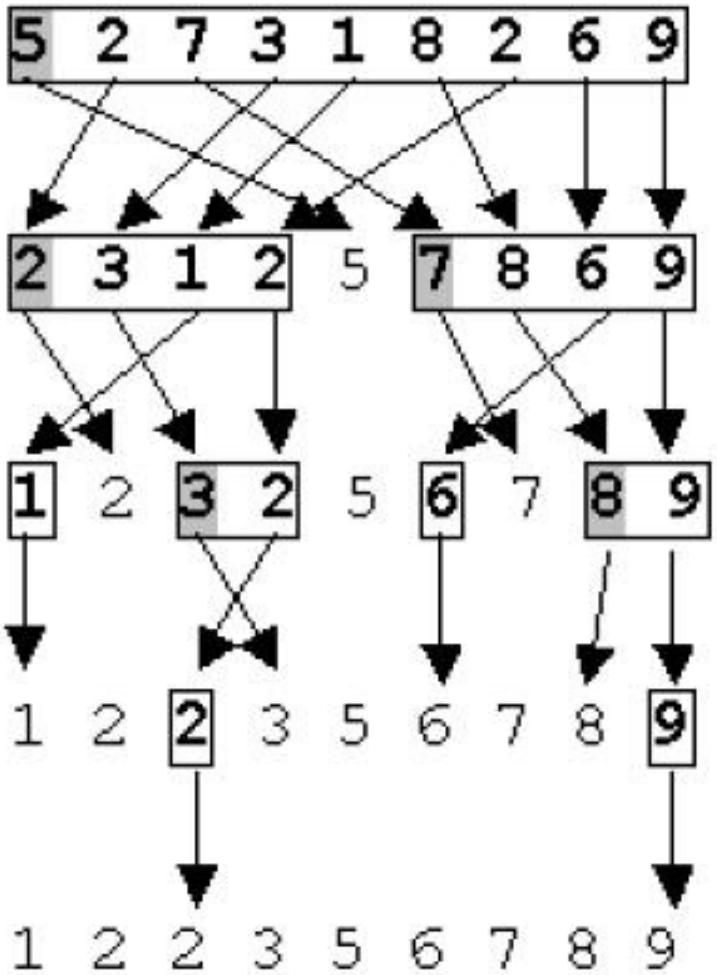
MÉTODO DE ORDENAMIENTO QUICKSORT



NÉSTOR ARIEL GONZÁLEZ HINOJOSA, NESTOR 22300859

EMMANUEL BUENROSTRO BRISEÑO, EMMANUEL 22300891

JOSE GUILLERMO OLVERA PALACIOS, MEMO 22300945

ESTRUCTURA DE DATOS I

KARLA ARELI ISAAC RODRIGUEZ

DESARROLLO DE SOFTWARE

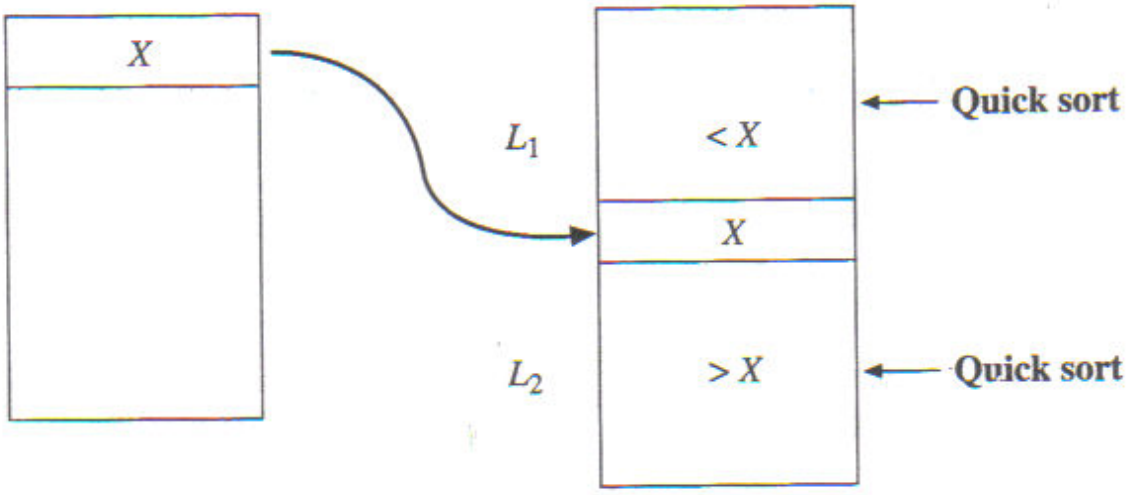
28/08/24

EQUIPO 4

INVESTIGACIÓN

QUICKSORT

El algoritmo quick sort se basa en la estrategia divide-y-vencerás, porque divide el problema en dos subproblemas, que se resuelven de manera individual e independiente. Los resultados se unen después. Dado un conjunto de números a1, a2,..., an, se escoge un elemento X para dividir dicho conjunto en dos listas, como se muestra a continuación.



Después de la división, el quick sort puede aplicarse en forma recursiva tanto a L1 como a L2, con lo cual se obtiene una lista ordenada, ya que L1 contiene a todos los a¡ menores que o iguales a X y L2 contiene a todos los ai mayores que X.

El problema es ¿cómo dividir la lista original? Ciertamente, no debe usarse X para recorrer toda la lista y decidir si un a¡ es menor que o igual a X. Hacer lo anterior provoca una gran cantidad de intercambio de datos. Sino que se utilizan dos apuntadores que se mueven al centro y realizan intercambios de datos según sea necesario.

El mejor caso de quick sort ocurre cuando X divide la lista justo en el centro. Es decir, X produce dos sublistas que contienen el mismo número de elementos. En este caso, la primera ronda requiere n pasos para dividir la lista original en dos listas. Para la ronda siguiente, para

cada sublista, de nuevo se necesitan n/2 pasos (ignorando el elemento usado para la división). En consecuencia, para la segunda ronda nuevamente se requieren 2 • (n/2) = n pasos. Si se supone que n = 2^P, entonces en total se requieren p*n pasos. Sin embargo, p = log2 n. Así, para el mejor caso, la complejidad temporal del quick sort es O(n log n).

El peor caso del quick sort ocurre cuando los datos de entrada están ya ordenados o inversamente ordenados. En estos casos, todo el tiempo simplemente se está seleccionando el extremo (ya sea el mayor o el menor). Por lo tanto, el número total de pasos que se requiere en el quick sort para el peor caso es:

n + (n - 1) + ••• + 1 = n/2 (n + 1) = O(n^2).

Para analizar el caso promedio, sea T(n) que denota el número de pasos necesarios para llevar a cabo el quick sort en el caso promedio para n elementos. Se supondrá que después de la operación de división la lista se ha dividido en dos sublistas. La primera de ellas contiene s elementos y la segunda contiene (n - s) elementos. El valor de s varía desde 1 hasta n y es necesario tomar en consideración todos los casos posibles a fin de obtener el desempeño del caso promedio. Para obtener T(n) es posible aplicar la siguiente fórmula:

T(n) = Promedio(T(s) + T(n - s)) + cn con l<=s<=n

donde cn denota el número de operaciones necesario para efectuar la primera operación de división. (Cada elemento es analizado antes de dividir en dos sublistas la lista original). Al resolver matemáticamente, se obtiene una eficiencia O(n log n).

De igual manera, se procede para hacer un análisis al algoritmo de Mergesort.

QuickSort, método de ordenamiento rápido. El método de ordenamiento QuickSort es actualmente el más eficiente y veloz de los métodos de ordenación interna. Este método es una mejora sustancial del método de intercambio directo y recibe el nombre de QuickSort por la velocidad con que ordena los elementos del arreglo. Es un algoritmo basado en la técnica de divide y vencerás, que permite, en promedio, ordenar “n” elementos en un tiempo proporcional a “n Log n”.

La idea central de este algoritmo consiste en lo siguiente: Se toma un elemento “x” de una posición cualquiera del arreglo. Se trata de ubicar a “x” en la posición correcta del arreglo, de tal forma que todos los elementos que se encuentran a su izquierda sean menores o iguales a “x” y todos los elementos que se encuentren a su derecha sean mayores o iguales a “x”. Se repiten los pasos anteriores pero ahora para los conjuntos de datos que se encuentran a la izquierda y a la derecha de la posición correcta de “x” en el arreglo. Reubicar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada. Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados. Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido.

¿Qué es el algoritmo QuickSort?

El algoritmo QuickSort, creado por Tony Hoare en 1959, es un eficiente y ampliamente utilizado algoritmo de ordenamiento que se basa en el principio de dividir y conquistar. A diferencia de MergeSort, que divide el arreglo en mitades iguales, QuickSort utiliza un pivote para dividir el arreglo en subarreglos.

El proceso comienza eligiendo un elemento como pivote y reordenando el arreglo de manera que todos los elementos menores que el pivote estén a su izquierda y todos los elementos mayores estén a su derecha. Luego, el algoritmo se aplica recursivamente a cada subarreglo generado.

El rendimiento del QuickSort depende en gran medida de la elección del pivote, pero en promedio tiene una complejidad de O(n log n). Sin embargo, en el peor de los casos, cuando el pivote siempre es el mínimo o el máximo, la complejidad puede degenerar a O(n^2). A pesar de esto, QuickSort sigue siendo ampliamente utilizado debido a su eficiencia y simplicidad de implementación.

Diferencias con MergeSort.

- *QuickSort:*
  1. Divide el arreglo utilizando un pivote.
  2. No tiene una etapa de combinación explícita.
  3. No es estable en su implementación básica.
  4. Puede degradarse a O(n^2) en el peor caso.
  5. Puede ser implementado "in-place".

- *MergeSort:*
  1. Divide el arreglo en mitades iguales.
  2. Tiene una etapa de combinación donde las sublistas se fusionan.
  3. Es estable.
  4. Siempre tiene complejidad O(n log n).
  5. Tiende a usar más memoria.

Pasos del algoritmo.

1. Elección del pivote: Selecciona un elemento del arreglo como pivote. Este elemento puede ser elegido de diversas maneras, como el primer elemento, el último elemento o un elemento al azar.
2. Partición: Reordena los elementos del arreglo de manera que todos los elementos menores que el pivote estén a su izquierda y todos los elementos mayores estén a su derecha. Esto se hace de tal manera que el pivote quede en su posición final en el arreglo ordenado. Este paso también divide el arreglo en dos subarreglos alrededor del pivote.
3. Recursión: Aplica recursivamente los pasos 1 y 2 a los subarreglos generados por la partición. Es decir, se repite el proceso de selección de pivote y partición en cada subarreglo hasta que el arreglo esté completamente ordenado. Este paso utiliza la naturaleza del paradigma "dividir y conquistar".
4. Combinación (implícita): No hay una etapa de combinación explícita en QuickSort, ya que los elementos se van ordenando "in-place" durante el proceso de partición.

El algoritmo trabaja de la siguiente forma:

Elegir un elemento del conjunto de elementos a ordenar, al que llamaremos pivote. Resituar los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada. La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha. Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados. Como se puede suponer, la eficiencia del algoritmo depende de la posición en la que termine el pivote elegido. En el mejor caso, el pivote termina en el centro de la lista, dividiéndola en dos sublistas de igual tamaño. En este caso, el orden de complejidad del algoritmo es $O(n \cdot \log n)$. En el peor caso, el pivote termina en un extremo de la lista. El orden de complejidad del algoritmo es entonces de $O(n^2)$. El peor caso dependerá de la implementación del algoritmo, aunque habitualmente ocurre en listas que se encuentran ordenadas, o casi ordenadas. Pero principalmente depende del pivote, si por ejemplo el algoritmo implementado toma como pivote siempre el primer elemento del array, y el array que le pasamos está ordenado, siempre va a generar a su izquierda un array vacío, lo que es ineficiente. En el caso promedio, el orden es $O(n \cdot \log n)$. No es extraño, pues, que la mayoría de optimizaciones que se aplican al algoritmo se centren en la elección del pivote.

Como se mencionó anteriormente, el algoritmo quicksort ofrece un orden de ejecución $O(n^2)$ para ciertas permutaciones "críticas" de los elementos de la lista, que siempre surgen cuando se elige el pivote «a ciegas». La permutación concreta depende del pivote elegido, pero suele corresponder a secuencias ordenadas. Se tiene que la probabilidad de encontrarse con una de estas secuencias es inversamente proporcional a su tamaño.

Los últimos pases de quicksort son numerosos y ordenan cantidades pequeña de elementos. Un porcentaje medianamente alto de ellos estarán dispuestos de una manera similar al peor caso del algoritmo, volviendo a éste ineficiente. Una solución a este problema consiste en ordenar las secuencias pequeñas usando otro algoritmo. Habitualmente se aplica el algoritmo de inserción para secuencias de tamaño menores de 8-15 elementos.

Pese a que en secuencias largas de elementos la probabilidad de hallarse con una configuración de elementos "crítica" es muy baja, esto no evita que sigan apareciendo (a veces, de manera intencionada). El algoritmo “introsort” es una extensión del algoritmo quicksort que resuelve este problema utilizando heapsort en vez de quicksort cuando el número de recursiones excede al esperado.

El método de QuickSort es ampliamente utilizado en diferentes áreas debido a su eficiencia. Aquí te dejo tres ejemplos de dónde se puede aplicar:

Bases de Datos: QuickSort es ideal para ordenar grandes cantidades de registros en bases de datos. Por ejemplo, si se necesita ordenar los registros de una tabla por un campo específico, como el nombre de usuario o la fecha de creación, QuickSort permite hacerlo de manera eficiente incluso cuando la cantidad de datos es considerable.

Sistemas de Archivos: En sistemas operativos, QuickSort se puede utilizar para ordenar archivos en un directorio por nombre, fecha de modificación, tamaño, etc. Esto ayuda a mejorar la rapidez con la que un usuario puede acceder a los archivos deseados en una estructura organizada.

Procesamiento de Imágenes: QuickSort se puede utilizar en algoritmos de procesamiento de imágenes para ordenar píxeles o valores de color. Por ejemplo, en la aplicación de filtros de mediana en imágenes, donde se requiere ordenar los valores de los píxeles en un área específica para determinar el valor de la mediana.

PRESENTACIÓN

INTRODUCCIÓN

El método Quick Sort, desarrollado por Tony Hoare en 1960, es actualmente uno de los algoritmos de ordenación interna más eficientes y rápidos, conocido también como método de ordenamiento por partición. Basado en la técnica de divide y vencerás, permite ordenar n elementos en un tiempo promedio proporcional a n log n, lo que lo convierte en una mejora sustancial del método de intercambio directo y en el algoritmo de ordenamiento más utilizado a nivel mundial.

¿EN QUÉ CONSISTE?

El método de ordenación Quicksort se basa en la estrategia de divide y vencerás, dividiendo un conjunto de elementos en dos sublistas mediante la elección de un pivote. Los elementos menores o iguales al pivote se colocan en la primera sublista, mientras que los mayores se ubican en la segunda. Luego, el algoritmo se aplica recursivamente a estas sublistas, reorganizándolas hasta que toda la lista esté ordenada. El mejor caso de Quicksort ocurre cuando el pivote divide la lista de manera equitativa, lo que lleva a una complejidad temporal de estable. Sin embargo, en el peor caso, cuando la lista ya está ordenada, la complejidad puede aumentar. El pivote es un elemento seleccionado que se usa como punto de referencia

¿EN DÓNDE SE USA?

QuickSort se utiliza en situaciones donde se requiere un algoritmo de ordenación rápido y eficiente para grandes volúmenes de datos. Es especialmente adecuado para aplicaciones donde la velocidad es crucial y no se necesita mantener el orden relativo de los elementos iguales. Su implementación recursiva lo hace ideal para arreglos que pueden dividirse fácilmente en listas más pequeñas, como en bases de datos, sistemas operativos y bibliotecas de software. También es útil en algoritmos que se benefician de una ordenación en el lugar, es decir, sin utilizar memoria adicional significativa.

COMPLEJIDAD

El rendimiento de QuickSort depende de la elección del pivote. En el mejor de los casos, cuando el pivote divide el arreglo de manera equitativa, el algoritmo muestra un excelente rendimiento siendo su complejidad de O(n log n). Por otro lado, en el peor de los casos, cuando el pivote es constantemente el valor extremo del arreglo, su eficiencia se degrada y su complejidad cambia siendo de o(n^2).

VENTAJAS

QuickSort es un algoritmo de ordenamiento muy eficiente y rápido, especialmente en el caso promedio, donde su complejidad temporal es O(n log n). Esto se debe a que permite dividir el problema en subproblemas más pequeños y manejables e ir juntando las soluciones hasta tenerla completa. Además, QuickSort es un algoritmo "in-place", lo que significa que no requiere espacio adicional significativo en memoria, haciendo un uso eficiente de los recursos disponibles.

DESVENTAJAS

En su peor caso, que ocurre cuando el pivote seleccionado no divide bien la lista, la complejidad temporal puede degradarse a $O(n^2)$. Esto sucede frecuentemente cuando los datos ya están ordenados o inversamente ordenados, lo que lo hace menos eficiente en estos escenarios. Además, QuickSort no es un algoritmo estable en su implementación básica, lo que significa que no necesariamente mantiene el orden relativo de los elementos iguales.

ITERACIONES

En total se hacen n iteraciones, debido a que todos los elementos son pivote alguna vez, y lo son máximo una vez.

Aunque muchas veces el elemento es pivote cuando es un solo valor (el mismo) por orden, entonces no se hace nada.

PREGUNTAS

1. ¿Cuál es la complejidad normalmente en un arreglo por el método QuickSort?

2. ¿Qué es un pivote?

3. ¿Para qué tipo de arreglos funciona mejor cuando su complejidad es poca?

4. ¿Cuál es una de las ventajas del método QuickSort?

5.¿Con qué pivote se obtiene el caso más eficiente?

R1. O(nlogn)

R2. El pivote es un elemento seleccionado que se usa como punto de referencia.

R3. Para arreglos con un gran volumen

R4. No necesita de almacenamiento extra/Es muy rápido y eficiente

R5. El valor del medio (tomando los valores, no la posición)

REFERENCIAS

Método quick sort. (s. f.). http://cidecame.uaeh.edu.mx/lcc/mapa/PROYECTO/libro9/mtodo_quick_sort.html

¿Qué es el algoritmo QuickSort? (2024, 1 abril). ASIMOV Ingeniería S. de R.L. de C.V. https://asimov.cloud/blog/programacion-5/que-es-el-algoritmo-quicksort-275

colaboradores de Wikipedia. (2024, 23 mayo). *Quicksort*. Wikipedia, la Enciclopedia Libre. https://es.wikipedia.org/wiki/Quicksort

https://www.udb.edu.sv/udb_files/recursos_guias/informatica-ingenieria/programacion-iv/2019/ii/guia-4.pdf

Sánchez, S. J., & Martinez, M. G. (2013). *Estructuras de Datos* (1.ª ed., Vol. 1) [Libro digital]. https://www.escom.ipn.mx/docs/oferta/matDidacticoISC2009/EDts/Libro_EstructuraDatos.pdf