QUICKSORT

QWERTY

Juan Cid Carreño Monica Riquelme Valery Soto Lastra
Gonzalo Zúñiga Palacios

Departamento de Informática
Universidad Tecnológica Metropolitana

Presentación sobre Metodos de Ordenamiento, 2013

Índice

Índice

1. Introducción
   - Definición
   - Descripción
2. Nuestros resultados
   - Explicación - Ejemplos
   - Resultados - Conclusiones

Prólogo

- Uno de los problemas mas comunes en la informática, es el manejo de grandes cantidades de datos.
- Surgen una gran cantidad de algoritmos de búsqueda y ordenamiento de dicha información.
- Así es como un algoritmo de búsqueda esta encargado de encontrar datos con características especificas.
- Y de ordenamiento para organizar está información con criterios específicos (principalmente numérico).

Prólogo

- Uno de los problemas mas comunes en la informática, es el manejo de grandes cantidades de datos.
- Surgen una gran cantidad de algoritmos de búsqueda y ordenamiento de dicha información.
- Así es como un algoritmo de búsqueda esta encargado de encontrar datos con características especificas.
- Y de ordenamiento para organizar está información con criterios específicos (principalmente numérico).

Prólogo

- Uno de los problemas mas comunes en la informática, es el manejo de grandes cantidades de datos.
- Surgen una gran cantidad de algoritmos de búsqueda y ordenamiento de dicha información.
- Así es como un algoritmo de búsqueda esta encargado de encontrar datos con características especificas.
- Y de ordenamiento para organizar está información con criterios específicos (principalmente numérico).

Prólogo

- Uno de los problemas mas comunes en la informática, es el manejo de grandes cantidades de datos.
- Surgen una gran cantidad de algoritmos de búsqueda y ordenamiento de dicha información.
- Así es como un algoritmo de búsqueda esta encargado de encontrar datos con características especificas.
- Y de ordenamiento para organizar está información con criterios específicos (principalmente numérico).

Estudio del Arte

- Algoritmo creado por Charles Antony Richard Hoare.
- Científico británico en computación.
- Creó este método en 1960 cuando visito la Universidad de Moscú.
- Siento estudiante intento traducir un diccionario de inglés para ruso, ordenando las palabras, siendo su objetivo reducir el problema original en subproblemas, que pudieran ser resueltos mas fácil y rápidamente.

Estudio del Arte

- Algoritmo creado por Charles Antony Richard Hoare.
- Científico británico en computación.
- Creó este método en 1960 cuando visito la Universidad de Moscú.
- Siento estudiante intento traducir un diccionario de inglés para ruso, ordenando las palabras, siendo su objetivo reducir el problema original en subproblemas, que pudieran ser resueltos mas fácil y rápidamente.

Estudio del Arte

- Algoritmo creado por Charles Antony Richard Hoare.
- Científico británico en computación.
- Creó este método en 1960 cuando visito la Universidad de Moscú.
- Siento estudiante intento traducir un diccionario de inglés para ruso, ordenando las palabras, siendo su objetivo reducir el problema original en subproblemas, que pudieran ser resueltos mas fácil y rápidamente.

Estudio del Arte

- Algoritmo creado por Charles Antony Richard Hoare.
- Científico británico en computación.
- Creó este método en 1960 cuando visito la Universidad de Moscú.
- Siento estudiante intento traducir un diccionario de inglés para ruso, ordenando las palabras, siendo su objetivo reducir el problema original en subproblemas, que pudieran ser resueltos mas fácil y rápidamente.

Definición Quicksort

- Es un algoritmo basado en la técnica “Divide y vencerás”.
- Permite en promedio, ordenar n elementos en un tiempo proporcional a $n * log(n)$.

Definición Quicksort

- Es un algoritmo basado en la técnica “Divide y vencerás”.
- Permite en promedio, ordenar n elementos en un tiempo proporcional a $n * log(n)$.

Índice

Como ejecuta Quicksort

- Elegir un elemento de la lista, al que se llama pivote.
- Resituar los otros elementos de la lista, a cada lado del pivote, donde a un lado queden los menores, y al otro lado los mayores.
- La lista queda dividida en dos sublitas, izquierda y derecha del pivote.
- Se repite este proceso, de manera recursiva, para cada sublista generada, mientras ellas contengan mas de un elemento.

Como ejecuta Quicksort

- Elegir un elemento de la lista, al que se llama pivote.
- Resituar los otros elementos de la lista, a cada lado del pivote, donde a un lado queden los menores, y al otro lado los mayores.
- La lista queda dividida en dos sublitas, izquierda y derecha del pivote.
- Se repite este proceso, de manera recursiva, para cada sublista generada, mientras ellas contengan mas de un elemento.

Como ejecuta Quicksort

- Elegir un elemento de la lista, al que se llama pivote.
- Resituar los otros elementos de la lista, a cada lado del pivote, donde a un lado queden los menores, y al otro lado los mayores.
- La lista queda dividida en dos sublitas, izquierda y derecha del pivote.
- Se repite este proceso, de manera recursiva, para cada sublista generada, mientras ellas contengan mas de un elemento.

Como ejecuta Quicksort

- Elegir un elemento de la lista, al que se llama pivote.
- Resituar los otros elementos de la lista, a cada lado del pivote, donde a un lado queden los menores, y al otro lado los mayores.
- La lista queda dividida en dos sublitas, izquierda y derecha del pivote.
- Se repite este proceso, de manera recursiva, para cada sublista generada, mientras ellas contengan mas de un elemento.

Complejidad Teórica

- Es el algoritmo de ordenación mas rápido. Su tiempo de ejecución promedio es $O(Nlog(N))$.
- Para el peor caso tiene un tiempo de $O(N^2)$, pero si se codifica correctamente, éste caso será improbable.
- En la práctica el hecho de que sea el mas rápido está dado por un ciclo interno muy ajustado (pocas operaciones).
- No requiere memoria adicional en su forma recursiva, pero de manera iterativa la necesita para la pila.
- No es estable.

Complejidad Teórica

- Es el algoritmo de ordenación mas rápido. Su tiempo de ejecución promedio es $O(Nlog(N))$.
- Para el peor caso tiene un tiempo de $O(N^2)$, pero si se codifica correctamente, éste caso será improbable.
- En la práctica el hecho de que sea el mas rápido está dado por un ciclo interno muy ajustado (pocas operaciones).
- No requiere memoria adicional en su forma recursiva, pero de manera iterativa la necesita para la pila.
- No es estable.

Complejidad Teórica

- Es el algoritmo de ordenación mas rápido. Su tiempo de ejecución promedio es $O(Nlog(N))$.
- Para el peor caso tiene un tiempo de $O(N^2)$, pero si se codifica correctamente, éste caso será improbable.
- En la práctica el hecho de que sea el mas rápido está dado por un ciclo interno muy ajustado (pocas operaciones).
- No requiere memoria adicional en su forma recursiva, pero de manera iterativa la necesita para la pila.
- No es estable.

Complejidad Teórica

- Es el algoritmo de ordenación mas rápido. Su tiempo de ejecución promedio es $O(Nlog(N))$.
- Para el peor caso tiene un tiempo de $O(N^2)$, pero si se codifica correctamente, éste caso será improbable.
- En la práctica el hecho de que sea el mas rápido está dado por un ciclo interno muy ajustado (pocas operaciones).
- No requiere memoria adicional en su forma recursiva, pero de manera iterativa la necesita para la pila.
- No es estable.

Complejidad Teórica

- Es el algoritmo de ordenación mas rápido. Su tiempo de ejecución promedio es $O(Nlog(N))$.
- Para el peor caso tiene un tiempo de $O(N^2)$, pero si se codifica correctamente, éste caso será improbable.
- En la práctica el hecho de que sea el mas rápido está dado por un ciclo interno muy ajustado (pocas operaciones).
- No requiere memoria adicional en su forma recursiva, pero de manera iterativa la necesita para la pila.
- No es estable.

Complejidad Teórica

- Tiempo de ejecución o caso promedio. La complejidad para dividir una lista de N es $O(n)$.Cada sub-lista genera en promedio dos sublistas mas de largo $n/2$. Por lo tanto la complejidad se define en forma recurrente como:
  - $f(1) = 1$
  - $f(n) = n + 2 * f(n/2)$
- La forma cerrada de esta expresión es: $f(n) = n * log(2n)$
- Es decir, la complejidad es $O(n * log(2n))$.

Complejidad Teórica

- Tiempo de ejecución o caso promedio. La complejidad para dividir una lista de N es $O(n)$.Cada sub-lista genera en promedio dos sublistas mas de largo $n/2$. Por lo tanto la complejidad se define en forma recurrente como:
  - $f(1) = 1$
  - $f(n) = n + 2 * f(n/2)$
- La forma cerrada de esta expresión es: $f(n) = n * log(2n)$
- Es decir, la complejidad es $O(n * log(2n))$.

Complejidad Teórica

- Tiempo de ejecución o caso promedio. La complejidad para dividir una lista de N es $O(n)$.Cada sub-lista genera en promedio dos sublistas mas de largo $n/2$. Por lo tanto la complejidad se define en forma recurrente como:
  - $f(1) = 1$
  - $f(n) = n + 2 * f(n/2)$
- La forma cerrada de esta expresión es: $f(n) = n * log(2n)$
- Es decir, la complejidad es $O(n * log(2n))$.

Complejidad Teórica

- Tiempo de ejecución o caso promedio. La complejidad para dividir una lista de N es $O(n)$.Cada sub-lista genera en promedio dos sublistas mas de largo $n/2$. Por lo tanto la complejidad se define en forma recurrente como:
  - $f(1) = 1$
  - $f(n) = n + 2 * f(n/2)$
- La forma cerrada de esta expresión es: $f(n) = n * log(2n)$
- Es decir, la complejidad es $O(n * log(2n))$.

Índice

Ejemplo

- Se considera la siguiente lista: 7-5-3-8-9-10-14-6-4-1.
- Supongamos que el elemento que se seleccionó al azar fue el 8, entonces nuestra lista se divide en dos partes, una con los mayores y una con los menores.
  - L1: 7-5-3-6-4-1
  - L2: 9-10-14
- Si trabajamos con L1, y elegimos el 4 al azar, entonces nuestra lista nuevamente se divide en dos partes, una con los mayores, y otra con los menores.
  - L1: 3-1
  - L2: 7-5-6

Ejemplo

- Se considera la siguiente lista: 7-5-3-8-9-10-14-6-4-1.
- Supongamos que el elemento que se seleccionó al azar fue el 8, entonces nuestra lista se divide en dos partes, una con los mayores y una con los menores.
  - L1: 7-5-3-6-4-1
  - L2: 9-10-14
- Si trabajamos con L1, y elegimos el 4 al azar, entonces nuestra lista nuevamente se divide en dos partes, una con los mayores, y otra con los menores.
  - L1: 3-1
  - L2: 7-5-6

Ejemplo

- Se considera la siguiente lista: 7-5-3-8-9-10-14-6-4-1.
- Supongamos que el elemento que se seleccionó al azar fue el 8, entonces nuestra lista se divide en dos partes, una con los mayores y una con los menores.
  - L1: 7-5-3-6-4-1
  - L2: 9-10-14
- Si trabajamos con L1, y elegimos el 4 al azar, entonces nuestra lista nuevamente se divide en dos partes, una con los mayores, y otra con los menores.
  - L1: 3-1
  - L2: 7-5-6

Ejemplo

- Se considera la siguiente lista: 7-5-3-8-9-10-14-6-4-1.
- Supongamos que el elemento que se seleccionó al azar fue el 8, entonces nuestra lista se divide en dos partes, una con los mayores y una con los menores.
  - L1: 7-5-3-6-4-1
  - L2: 9-10-14
- Si trabajamos con L1, y elegimos el 4 al azar, entonces nuestra lista nuevamente se divide en dos partes, una con los mayores, y otra con los menores.
  - L1: 3-1
  - L2: 7-5-6

Ejemplo

- Se considera la siguiente lista: 7-5-3-8-9-10-14-6-4-1.
- Supongamos que el elemento que se seleccionó al azar fue el 8, entonces nuestra lista se divide en dos partes, una con los mayores y una con los menores.
    - L1: 7-5-3-6-4-1
    - L2: 9-10-14
- Si trabajamos con L1, y elegimos el 4 al azar, entonces nuestra lista nuevamente se divide en dos partes, una con los mayores, y otra con los menores.
    - L1: 3-1
    - L2: 7-5-6

Ejemplo

- Así sucesivamente hasta que se llega a una lista con dos elementos, los cuales se comparan y se retornan en orden.
- Lo mismo sucede con la L2 anterior (9-10-14), hasta que se obtiene la lista con dos elementos los cuales se ordenan y se retornan.
- Al finalizar, ambas listas estarán ordenadas (tanto la que tenía los mayores como la que tenía los menores que la llave seleccionada) y la lista completa estará ordenada.

Ejemplo

- Así sucesivamente hasta que se llega a una lista con dos elementos, los cuales se comparan y se retornan en orden.
- Lo mismo sucede con la L2 anterior (9-10-14), hasta que se obtiene la lista con dos elementos los cuales se ordenan y se retornan.
- Al finalizar, ambas listas estarán ordenadas (tanto la que tenía los mayores como la que tenía los menores que la llave seleccionada) y la lista completa estará ordenada.

Ejemplo

- Así sucesivamente hasta que se llega a una lista con dos elementos, los cuales se comparan y se retornan en orden.
- Lo mismo sucede con la L2 anterior (9-10-14), hasta que se obtiene la lista con dos elementos los cuales se ordenan y se retornan.
- Al finalizar, ambas listas estarán ordenadas (tanto la que tenía los mayores como la que tenía los menores que la llave seleccionada) y la lista completa estará ordenada.

Índice

Experimento - Mediciones

- Código (cpp y ruby)
- Medición

Experimento - Mediciones

- Código (cpp y ruby)
- Medición

Mejor Caso

- El mejor de los casos sucede cuando el pivote que escogemos, divide el arreglo en dos partes iguales en cada paso.
- Asi pues, tenemos $k = n/2$ y $n - k = n/2$de la matriz original de tamaño n.
- O(n).

Mejor Caso

- El mejor de los casos sucede cuando el pivote que escogemos, divide el arreglo en dos partes iguales en cada paso.
- Asi pues, tenemos $k = n/2$ y $n - k = n/2$de la matriz original de tamaño n.
- O(n).

Mejor Caso

- El mejor de los casos sucede cuando el pivote que escogemos, divide el arreglo en dos partes iguales en cada paso.
- Asi pues, tenemos $k = n/2$ y $n - k = n/2$de la matriz original de tamaño n.
- O(n).

Caso Promedio

- Para el caso promedio es mas complejo, y se necesitan herramientas matemáticas avanzadas.
- O(n^2).

Caso Promedio

- Para el caso promedio es mas complejo, y se necesitan herramientas matemáticas avanzadas.
- O(n^2).

Peor Caso

- Ahora se analizará el caso, cuando el pivote resultó ser el menor elemento de la matriz, de modo que tuvimos $k = 1$ y $n - k = n - 1$.
- En tal caso, tenemos $T(n) = T(1) + T(n-1) + \alpha n$.
- A continuación analizaremos el tiempo de complejidad de quicksort:

Peor Caso

- Ahora se analizará el caso, cuando el pivote resultó ser el menor elemento de la matriz, de modo que tuvimos $k = 1$ y $n - k = n - 1$.
- En tal caso, tenemos $T(n) = T(1) + T(n-1) + \alpha n$.
- A continuación analizaremos el tiempo de complejidad de quicksort:

Peor Caso

- Ahora se analizará el caso, cuando el pivote resultó ser el menor elemento de la matriz, de modo que tuvimos $k = 1$ y $n - k = n - 1$.
- En tal caso, tenemos $T(n) = T(1) + T(n-1) + \alpha n$.
- A continuación analizaremos el tiempo de complejidad de quicksort:

Peor Caso

- $T(n) = T(n-1) + T(1) + \alpha n$
- $= [T(n-2) + T(1) + \alpha(n-1)] + T(1) + \alpha n$
- $= T(n-2) + 2T(1) + (n-1+n)$
- $= [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1+n)$
- $= T(n-3) + 3T(1) + \alpha(n-2+n-1+n)$
- $= [T(n-4) + T(1) + \alpha(n-3)] + 3T(1) + \alpha(n-2+n-1+n)$
- $= T(n-4) + 4T(1) + \alpha(n-3+n-2+n-1+n)$
- $= T(n-i) + iT(1) + \alpha(n-i+1+...+n-2+n-1+n)$
- $= T(n-i) + iT(1) + \alpha(n-j)T(n) = T(1) + (n-1)T(1) + \alpha$
- $= nT(1) + \alpha(n * \frac{(n-2)-(n-2)(n-1)}{2})$
- $\sum_{j=0}^{i-2} j = \sum_{j=1}^{i-2}$

Conclusiones

- Este algoritmo posee una fácil implementación lo cual lo hace una buena elección para muchas aplicaciones.
- Trabaja muy bien con diferentes tipos de datos de ingreso, aparte de que usa menos recursos que los otros algoritmos de ordenamiento.
- Quicksort en el mejor de sus casos su tiempo es del orden $O(nlog_2(n))$ y en el peor es $O(n^2)$.

Conclusiones

- Este algoritmo posee una fácil implementación lo cual lo hace una buena elección para muchas aplicaciones.
- Trabaja muy bien con diferentes tipos de datos de ingreso, aparte de que usa menos recursos que los otros algoritmos de ordenamiento.
- Quicksort en el mejor de sus casos su tiempo es del orden $O(nlog_2(n))$ y en el peor es $O(n^2)$.

Conclusiones

- Este algoritmo posee una fácil implementación lo cual lo hace una buena elección para muchas aplicaciones.
- Trabaja muy bien con diferentes tipos de datos de ingreso, aparte de que usa menos recursos que los otros algoritmos de ordenamiento.
- Quicksort en el mejor de sus casos su tiempo es del orden $O(nlog_2(n))$ y en el peor es $O(n^2)$.

Bibliografía I

- http://es.wikipedia.org/wiki/Quicksort
- http://www.scribd.com/doc/54663992/INFORME-Quicksort
- http://upcanalisisalgoritmos.wikispaces.com/file/view/quicksort.pdf
- http://saforas.wordpress.com/2008/01/05/codigo-c-ordenamiento-quick-sort/