Documentación Práctica 6.
Estructura de Datos Lineales
Algoritmos iterativos de búsqueda y ordenamiento.
Quicksort.

Introducción.

Como se vio en la práctica pasada, práctica 5, el algoritmo de ordenamiento “Quicksort” es uno de los algoritmos de ordenamiento más eficaces que existen. En la documentación anterior estudiamos una manera en la que los elementos de una fila se organizan con respecto a un pivote, observamos que requeríamos recorrer toda la lista en busca de todo aquel elemento que fuera menor al pivote, extraerlo de la fila y posteriormente insertarlo antes del pivote y al finalizar todo el recorrido nos quedamos con dos sub-listas cuyos apuntadores de inicio y fin guardamos en una pila para posteriormente trabajar con ellos, y esto hasta que la pila quedará vacía lo cual indicaba que la lista había sido ordenada.

En esta versión del algoritmo “Quicksort” veremos una manera más eficiente de colocar todo aquel número menor al pivote antes de este y todo elemento mayor después.

Para esto hacemos uso de dos auxiliares un auxiliar que inicia el recorrido de la lista un elemento después de la fila y otro auxiliar que inicia el recorrido desde el final de la fila. Al primero lo llamaremos “a” y al segundo “A”. El algoritmo consiste en avanzar “a” o “A”, depende del programador, la condición para que “a” avance es que este sea menor al pivote y se detendrá cuando el elemento en el que está posicionado “a” sea mayor al pivote. La condición para que “A” avance es que este sea mayor al pivote y se detendrá cuando el elemento en el que se encuentra “A” se menor al pivote, cuando estos dos dejen de avanzar se realizara un cambio de posición entre “a” y “A”, este proceso se repite hasta que no haya elementos que intercambiar, hay un caso en especifico que nos permite posicionar al pivote en su posición correcta y este se da cuando la posición de “a” se encuentra por arriba de la de “A”, en este caso se hace el intercambio entre “a” y el pivote, con esto todos los elementos menores al pivote se encuentran a la izquierda o por debajo y todos los mayores al pivote se encuentran a la derecha o por arriba.

Este proceso genera dos sub-filas por lo cual guardaremos en un NodoP, como se mencionó en la práctica pasada es un nodo que guarda apuntadores a nodos, los apuntadores de inicio y fin de cada sub-fila en una fila y el procedimiento se repetirá hasta que la pila este vacía lo cual es indicación de que la fila ha sido ordenada.

Desarrollo.

A continuación, se mostrará el pseudocódigo versión número dos del algoritmo “Quicksort”, todos lo métodos usados en la práctica 5 fueron implementados en el desarrollo de esta versión del algoritmo. Por lo cual no se mostrarán en esta parte de la documentación.

Pseudocódigo ordenamiento “Quicksort” versión número dos.

- Entrada: I
  Salida: I
  Funcion Vacia quickSort(I)
    Llama FILA Fila //Llamada a clase FILA se instancia Fila
    Llama PILA Pila //Llamada a clase PILA se instancia Pila

    Inicio <- I
    Si(Inicio) entonces
      Fila.InicioFila(Inicio) //Llamada a función establecer inicio fila.
      Fila.FinFila(); //Llamada a función establecer final fila.
      Fin <- Fila.ObtenerFinal //Llamada a función obtener final fila.
      Puntero <- Fila.CrearNodoP(Inicio, Fin) //Llamada a función crear NodoP.
      Pila.PUSH(Puntero) //Llamada a función Push de pila.
      Mientras (Pila <> 0) Hacer
        Puntero <- Pila.POP //Llamada a función POP de pila.
        fila <- puntero.Inicio
        Fin <- puntero.Final
        Pivote <- fila
        a <- fila.Sig
        A <- Fin
        Si (fila.Sig) entonces
          Bandera <- verdadero
        Otro
          Bander <- Falso
        Fin Si
        Mientras (Bandera) Hacer
          Mientras (a.Dato < pivote.Dato y a <> Fin) hacer
            a <- a.Sig
          Fin Mientras
          Mientras (A.Dato >= pivote.Dato y A <> fila) hacer
            A<- A.Sig
          Fin Mientras
          Si (A <> pivote y A.Sig <> a y a.Dato >= pivote.Dato) entonces
            Si (A = Fin) entonces
              Fin <- a
            Fin Si
            Si (a.Sig = A) entonces
              a.Sig <- A.Sig
              Si(A.Sig) entonces
                (A.Sig).Ant <- a
              Fin Si
              A.Ant <- a.Ant
              (a.Ant).Sig <- A
              A.Sig <- a
              a.Ant <- A
            Otro
              Aux <- a.Sig
              a <- Fila.Extraer(a) //Llamada a función extraer.
              Fila.InsertarAntesPivote(Aux, a) //Llamada a función insertar antes de pivote.
              A <- Fila.Extraer(A) //Llamada a función extraer.

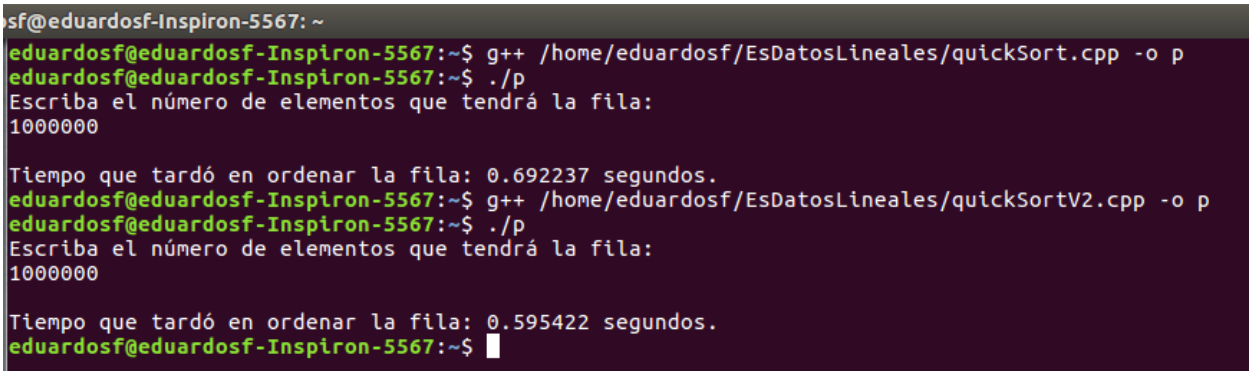
```
            Fila.InsertarAntesPivote(Aux, A)
            A <- a
          Otro
            Bandera <- Falso
          Fin Si
          Si (A.Sig = a y pivote <> A) entonces
            fila <- fila.Sig
            pivote <- Fila.Extraer(pivote) //Llamada a función extraer.
            Fila.InsertarAntesPivote(a, pivote) //Llamada a función insertar antes de pivote.
            Bandera <- Falso
          Otro
            Si (a = Fin y a.Dato < Pivote.Dato) entonces
              fila <- fila.Sig
              pivote <- Fila.Extraer(pivote) //Llamada a función extraer.
              pivote.Sig <- Fin.Sig
              Si (Fin.Sig) entonces
                (Fin.Sig).Ant <- pivote
              Fin Si
              Fin.Sig <- pivote
              pivote.Ant <- Fin
              Fin <- pivote
            Fin Si
        Fin Mientras
        Si (fila <> Fin) entonces
          Si (fila.Sig <> Fin) entonces
            Si (pivote <> fila) entonces
              puntero <- Fila.CrearNodoP(fila, pivote.Ant) //Llamada a función crear nodoP.
              Pila.PUSH(puntero) //Llamada a función PUSH de pila.
            Fin Si
            Si (pivote <> Fin) entonces
              puntero <- Fila.CrearNodoP(pivote.Sig, Fin) //Llamada a función crear nodoP.
              Pila.PUSH(puntero) //Llamada a función PUSH de pila.
            Fin Si
          Fin Si
      Fin Mientras
    Fin Si
    I <- Fila.InicioFila() //Llamada a función establecer inicio fila.
  Fin Procedimiento
```

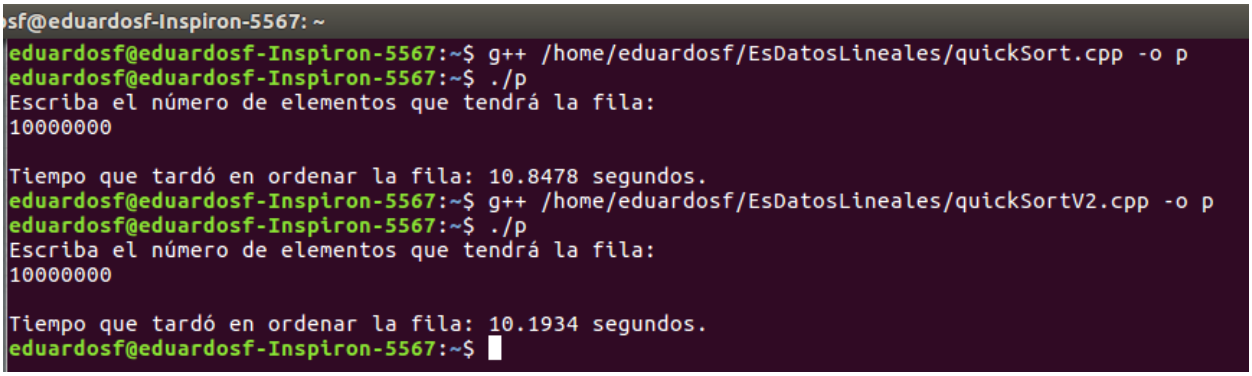
Resultados.

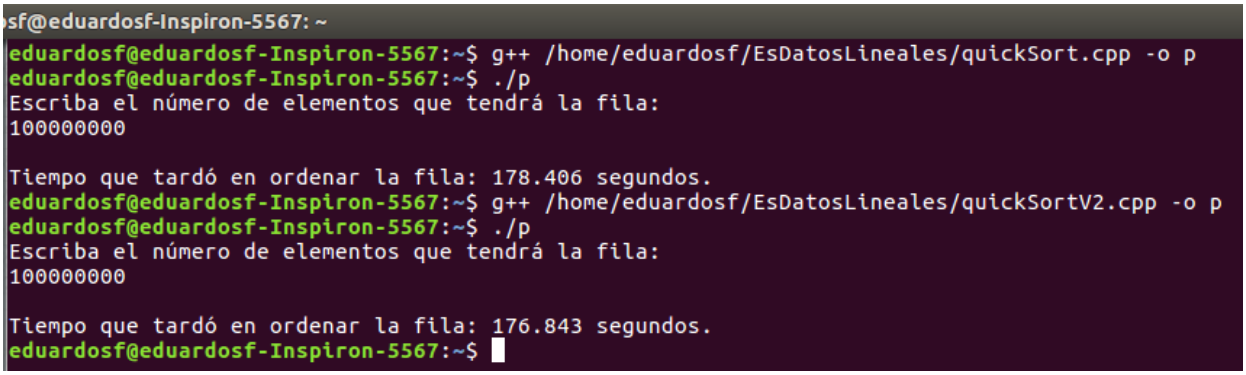
Con un ciclo “para” se crean uno por uno los nodos con su respectivo número aleatorio y se hace uso del método que inserta nodos en la fila, esto hasta que terminé de crear la cantidad de nodos que el usuario indicó, ya que al ejecutar el programa aparecerá en consola la indicación al usuario de proporcionar la cantidad de nodos que tendrá la fila. Ya con todos los nodos creados e insertados en la fila se hará uso del método quickSort, el cual ordenará los nodos de manera ascendente. Finalizada la ejecución del método aparecerá en consola el tiempo en segundos que le tomó al método en realizar el ordenamiento.

En esta versión del algoritmo al igual que el anterior, también se le dará al usuario o a quien revise el código la opción de decidir si requiere imprimir la fila o no. Como se mencionó en la práctica pasada, esto se recomienda si el número de elementos es menor a mil pues así mostrará al menos una porción de estos elementos ordenados. Si se desea hacer deberá descomentar las líneas 260, 261, 267 y 268 del código.

A continuación, se mostrarán una serie de imágenes que nos demuestran la efectividad de este algoritmo al ordenar una gran cantidad de elementos y se comparará con la versión del algoritmo “Quicksort” de la práctica 5.







Como se puede observar el tiempo que le toma a la primera versión del algoritmo “Quicksort” es ligeramente más lenta que la segunda versión del algoritmo “Quicksort”, esto es más evidente cuando la cantidad de elementos por ordenar es mucho más grande. Aunque la diferencia parece mínima al ordenar 100 millones de elementos, se puede observar que si el número de elementos es mayor la diferencia de tiempo entre estos dos métodos se hará más grande.

Manual de Usuario:

- Para compilar el programa es necesario escribir lo siguiente en la terminal de LINUX:
  - g++ /home/nombreUsuario/ubicaciónArchivo/nombreArchivo.cpp -o p
  - Siendo “nombreArchivo” el nombre que lleva el programa, en este caso lleva por nombre: quickSortV2
  - Si la ubicación del archivo se encuentrá en descargas la línea para compilar se vería de la siguiente manera:
    - g++ /home/nombreUsuario/Downloads/quickSortV2.cpp -o p
- Para ejecutar el programa se introduce lo siguiente:
  - ./p
  - La última letra de la línea de compilación indica la letra que va después del “ ./ ”.