Universidad Nacional Autónoma de México

EDA II Profesor: Tista García Edgar

Realizado por el equipo 4:

Argüello Dante

Gaytán Nava Aarón Emmanuel

Marco

**Marco Teórico**

**Análisis**

* **File Manager:** Para el manejo de todos los archivos se creó la clase FileManager, la cual posee distintos métodos para poder crear archivos, leer los archivos y escribir dentro de los archivos.
  + Método **readBlockFile(String fileName)**: Éste método nos permite leer el archivo que contiene a nuestras claves o a los auxiliares *por bloques,* el bloque queda definido por el atributo “n” de nuestro objeto creado, este atributo es el que deberá ser modificado para poder cambiar el tamaño de los bloques a extraer, el parámetro fileName corresponde al nombre del archivo donde se encuentran las llaves. Este método utiliza el atributo de pivote para saber si ha terminado de leer los archivos y/o saber en que elemento del archivo se quedó leyendo. Regresa una lista de tipo String que contiene todas las claves leídas de ese bloque.
  + Método **readKeyFile(String fileName)**: Éste método lee completamente un archivo que contiene a nuestras claves o a los auxiliares, el parámetro fileName corresponde al nombre del archivo donde se encuentran las llaves. Regresa una lista de tipo String que contiene todas las claves leídas.
  + Método **countKeySize(String fileName):** Éste método cuenta el número de claves contenidas en nuestro archivo, el parámetro fileName corresponde al nombre del archivo donde se encuentran las llaves. Regresa un número entero que corresponde al número de claves contenidas en nuestro archivo.
  + Método **writeKeyFile(String fileName, String keyToWrite, boolean reWrite):** Éste método es el encargado de escribir o sobrescribir en el archivo deseado, el parámetro fileName corresponde al nombre del archivo donde se desea escribir, el parámetro keyToWrite corresponde a la String o clave que deseamos escribir en el archivo y el parámetro reWrite nos indica si se va a sobrescribir el archivo o la cadena se va a escribir al final de nuestro archivo.
  + Método **createFile(String fileName):** Crea un archivo, el parámetro fileName corresponde al nombre del archivo que se desea crear.
* **Sort Polyphase**: Esta clase es la encargada de realizar el ordenamiento por polifase, lo primero que hace es crear un objeto de tipo FileManager y crear los archivos correspondientes a los archivos auxiliares (3), seguido de esto crea un archivo llamado fileIT que es donde se verán reflejadas todas las iteraciones de nuestro algoritmo. Seguido de esto empieza a leer los bloques de tamaño n indicado en el parámetro del constructor del objeto.

Después de esto realiza el ordenamiento del primer bloque de elementos, dicho ordenamiento utiliza un algoritmo de ordenamiento interno, para nuestro caso es QuickSort. Para el ordenamiento interno se utiliza una clase llamada KeyUtilities en el paquete de Utilities:

* + **KeysUtilities**: Es una clase que realiza el ordenamiento interno de la clase su único método es **sortBlock(List<String> keys, String fileName),** el cual recibe la lista de claves a ordenar y el nombre del archivo auxiliar en el que se escribirán, dentro de éste método las claves se ordenan utilizando la clase QuickSort.

Después de terminar este primer ordenamiento se procede a leer nuevamente los bloques creados en los archivos auxiliares y fusionarlos a través de un método de intercalación implementado en la clase KeysIntercalation en el paquete de Utilities:

* + **KeysIntercalation:** Es la clase que realiza la intercalación de las llaves contenidas en dos archivos para ello usa dos métodos el primero y más importante es **intercalation(List<String> keyOne, List<String> keyTwo, String fileName),** sus parámetros keyOne y keyTwo son las dos listas que contienen las claves a intercalar dichas listas fueron creadas previamente con la lectura de los dos archivos auxiliares, el parámetro fileName corresponde al nombre del archivo donde vamos a ingresar nuestro nuevo bloque de claves ordenado, lo único que hace éste método es ir comparando el primer elemento de amas listas e introducir en una nueva lista el elemento de menor valor lexicográfico, esto hasta que las dos listas queden vacías y la tercer lista contenga las claves ordenadas, por último imprime dicha lista final gracias al método **printArray(String fileName, List<String> keys),** el cual recibe dicha lista y la imprime clave por clave en el archivo auxiliar ingresado en el parámetro fileName.

Dicho proceso de lectura e intercalación se realiza hasta que el tamaño del bloque sea mayor o igual al numero de claves leídas al principio. Cumplido este caso significa que nuestro archivo ha quedado ordenado. Es importante mencionar que para elegir la parte de nuestra String a ordenar se usa la clase **StringUtilities** contenida en nuestro paquete **Utilities**, la cual convierte nuestra clave completa en nombre, apellido o número de cuenta para su comparación. Al finalizar el ordenamiento nuestro programa nos indica donde esta ahora nuestro archivo con las claves ordenadas y *en caso de que haya cambiado de nuestro archivo original deberá salir y elegir el nuevo archivo* que contiene las claves, además todas las iteraciones se guardan en fileIT.