ALGORITMO LZW

El algoritmo LZW es un algoritmo de compresión adaptativa, es decir va leyendo el flujo de datos y en base a esto va ajustando el diccionario de manera que el diccionario se va generando y va comprimiendo mientras que se va leyendo el archivo, gracias a esto únicamente hay que leer el archivo una sola vez. Cabe comentar que el diccionario no es transmitido con el texto comprimido, ya que el descompresor puede reconstruirlo usando la misma lógica con que lo hace el compresor y si esta codificado de manera correcta, tendrá exactamente las mismas cadenas que el diccionario del compresor tenía.

Nota:Una vez hecha la compresión no tenemos letras sino que tenemos únicamente números los cuales pertenecen a las respectivas entradas del diccionario.

Implementación del algoritmo

Para implementar el algoritmo he hecho una clase la cual se llama LZW en ella están las creadoras para crear un objeto con el patrón single y cuatro métodos, uno no lo utilizo ya que es un método que se hereda de la clase Algoritmo y solo se utiliza para el algoritmo jpeg, otro devuelve el nombre del algoritmo y en los otros dos métodos esta implementada la compresión y descompresión del algoritmo.

En ambos métodos se pasa por parámetro un objeto de tipo MyByteCollection (este objeto es de una clase que hemos creado, la cual tiene dos atributos uno es un ArrayList<Byte> y el otro es un entero llamado índice ambos son atributos privados, los métodos que tiene esta clase son: ArrayList<Byte> getContenido(), void writeByte(Byte b), void writeIntetger(Integer v), void writeShort(Short v), byte readByte(), int readInt(), short readShort(), int readUnsignedShort(), int getSize(), void set Indice(int indice). Aparte de su creadora).

En el método **comprimir** para implementar la compresión he utilizado las siguientes estructuras de datos: un Map<String, Short> (en este map se cargan todos los elementos ascii, para poder realizar la compresión) un MyByteCollection (el MyByteCollection que se pasa por parámetro se utiliza para leer el contenido que se quiere comprimir y después se crea otro objeto de este tipo para guardar la compresión utilizando algunos de los métodos de la esta clase). La función retorna un MyByteCollection donde se encuentra el texto comprimido.

En el método **descomprimir** para implementar la descompresión he utilizado las siguientes estructuras de datos: un MByteCollection (el MyByteCollection que se pasa por parámetro se utiliza para leer el contenido que se quiere comprimir), un Map< Short,String> (en este map se va reconstruyendo el diccionario para poder ir haciendo la descompresión a medida que se va leyendo los datos comprimidos), ArrayList<String> (este se utiliza para ir guardando el texto descomprimido y para poder pasar el texto descomprimido a bytes, es decir pasar el ArrayList<String> a un ArrayList<Byte>, ya que queremos que un ArrayList<Byte>), ArrayList<Byte> (este se utiliza para guardar el texto descomprimido en bytes). La función retorna un ArrayList<Byte> donde se encuentra el texto descomprimido.

BÚSQUEDA DE INFORMACIÓN

Para la hacer la búsqueda de información he mirado videos y páginas en Internet para estar lo mejor informado posible, para poder hacer un buen algoritmo LZW, y también he tenido que mirar páginas sobre java, ya que no sabía programar bien en este lenguaje de programación, algunas de las páginas y vídeos son los siguientes (no están todas, ya que no fui guardando las páginas y videos que visitaba):

https://es.wikipedia.org/wiki/LZW

https://www.youtube.com/watch?v=8Hxo1bPPgCU

https://www.youtube.com/watch?v=fiAReFNIrKc

https://www.youtube.com/watch?v=5vlaSmPmE98

https://es.ccm.net/contents/731-compresion-lzw

https://javadesdecero.es/clases/string/

RESULTADO DE LOS JUEGOS DE PRUEBAS

He creado dos juegos de prueba con los nombre juego_de_prueba1.txt y juego_de_prueba2.txt, el primer juego de prueba (juego_de_prueba1.txt) tiene la finalidad de probar que el algoritmo funcione sin caracteres especiales y el segundo juego de prueba (juego_de_prueba2.txt) tiene la finalidad de probar que el algoritmo funcione con caracteres especiales.

El resultado del primer juego de prueba lo pasa de manera exitosa.

En segundo juego de prueba el algoritmo no funciona de manera correcta, por alguna razón que desconozco, solo sé que es debido a algunos caracteres especiales. Te comento la situación el algoritmo cuando va a comprimir carga todos los caracteres ascii en el diccionario el cuál se utiliza posteriormente para hacer la compresión, cuando va a consultar el código que le pertenece a un carácter especial peta y no tiene sentido porque primeramente cargo absolutamente todos los valores en ascii.

También el algoritmo tiene un problema en el tipo que utiliza para comprimir el archivo, es decir cuando va leyendo cada carácter del texto y va comprimiendo cada cadena de caracteres esta cadena se comprime en un número de tipo short cuando lo debería de hacer a nivel de bits. He invertido una cantidad de horas considerables mirando en foros como stackoverflow y varios del estilo y varias páginas para poder implementarlo a nivel de bits, y no ha habido manera, al final he tenido que dejarlo para poder hacer el resto de cosas que quedaban por hacer del trabajo.