Índice

Estructuras de Datos & Algoritmos

LZSS

Compresión

Estructuras de Datos Para la ventana deslizante que contiene los últimos n caracteres leídos del fichero usamos un ArrayList ya que es modificable su tamaño y también permite recorrerlo para encontrar coincidencias.

Para guardar los caracteres que tienen coincidencia hasta el momento hemos usado también un ArrayList por las misma razones que la ventana deslizante.

Algoritmos El único algoritmo que usamos en esta implementación de LZSS es Knuth-Morris-Pratt ya que aumenta un poco la eficiencia para encontrar coincidencias de un patrón dentro de la ventana deslizante. Lo hemos adaptado para que devuelva el indice empezando por el final de la primera ocurrencia del patrón, y si el patrón no tiene ocurre en la ventana simplemente retorna -1.

Descompresión

Estructuras de Datos Al igual que en la compresión y por la mismas razones usamos un ArrayList para guardar la ventana deslizante.

Algoritmos Para la implementación de la descompresión no usamos ningún algoritmo auxiliar de importancia.

LZ78

Compresión

Estructuras de Datos Para comprimir en el algoritmo LZ78 hemos utilizado un HashMap con key String y value Integer, actuando de diccionario y de manera que todos los caracteres que se leen del archivo input quedan guardados en este. En caso que ya se hayan guardado se guarda un string concatenando la entrada del diccionario con el siguiente carácter leído. Como archivo comprimido resultante se obtiene un conjunto de sets compuestos por un numero y en carácter.

Descompresión

Estructuras de Datos Para descomprimir hemos utilizado en este caso un HashMap inverso al de compresión, con key Integer y value String, actuando también como diccionario. Al descomprimir se lee primero un numero, que indica una entrada del diccionario y a continuación se lee el carácter del que va acompañado dicho numero.

Finalmente se escribe en el archivo de descompresión la concatenación del value correspondiente a la entrada del diccionario junto con el char leídos.

LZW

Compresión

Estructuras de Datos Para comprimir en el algoritmo LZW hemos utilizado un HashMap con key String y value Integer, actuando de diccionario y de manera que todos los caracteres que se leen del archivo input quedan guardados en este. El diccionario se ha de inicializar con los valores unitarios esperados en el archivo a comprimir y a medida que se vaya leyendo la entrada se irán añadiendo posibles combinaciones de éstos.

Descompresión

Estructuras de Datos Para descomprimir hemos utilizado en este caso un HashMap inverso al de compresión, con key Integer y value String, actuando también como diccionario. Al igual que en la compresión el diccionario se ha de inicializar con los valores unitarios esperados en el archivo a descomprimir y a medida que se vaya leyendo la entrada se irán añadiendo posibles combinaciones de éstos.

LZW

Compresión

Estructuras de Datos Para comprimir en el algoritmo LZW hemos utilizado un HashMap con key String y value Integer, actuando de diccionario y de manera que todos los caracteres que se leen del archivo input quedan guardados en este. El diccionario se ha de inicializar con los valores unitarios esperados en el archivo a comprimir y a medida que se vaya leyendo la entrada se irán añadiendo podibles combinaciones de éstos.

Descompresión

Estructuras de Datos Para descomprimir hemos utilizado en este caso un HashMap inverso al de compresión, con key Integer y value String, actuando también como diccionario. Al igual que en la compresión el diccionario se ha de inicializar con los valores unitarios esperados en el archivo a descomprimir y a medida que se vaya leyendo la entrada se irán añadiendo podibles combinaciones de éstos.

JPEG

Compresión

Estructuras de Datos Para la compresión se ha usado una tabla Huffman con valores predefinidos.

Algoritmos

- *DCT (Discrete Cosine Transform)* Transforma los datos de la imagen como sumas de cosenos bidimensionales.
- *RLE (Run Length Encoding)* Codifica los datos según el número de 0 que lo preceden (Run) y los bits necesarios para representar-lo (Length). Los valores de Run y length son los que se codifican con la tabla Huffman. Gracias a la DCT, hay muchos 0 lo que permite comprimir los datos con RLE.

Descompresión

Estructuras de Datos Para descomprimir el archivo, se ha usado la misma tabla de Huffman que en la compresión pero en formato de árbol.

Algoritmos

- *inverse DCT (Discrete Cosine Transform)* Deshace la DCT de la compresión
- *RLE (Run Length Encoding)* Deshace el RLE de la compresión.